Arduino Básico

Arduino Básico

Michael McRoberts

Novatec

Editor: Rubens Prates
Tradução: Rafael Zanolli
Revisão gramatical: Marta Almeida Sá
Revisão técnica: Edgard Damiani
Editoração eletrônica: Camila Kuwabata / Carolina Kuwabata

ISBN: 978-85-7522-274-4

Histórico de impressões:

Janeiro/2012 Primeira reimpressão
Setembro/2011 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
E-mail: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

McRoberts, Michael
Arduino básico / Michael McRoberts ; [tradução Rafael Zanolli]. -- São Paulo : Novatec Editora, 2011.

Título original: Beginning arduino
ISBN 978-85-7522-274-4

1. Arduino (Linguagem de programação para computadores) 2. Eletrônicos - Processamento de dados I. Título.

11-05551 CDD-005.133

Índices para catálogo sistemático:

1. Arduino : Linguagem de programação : Computadores : Processamento de dados 005.133

VC20111215

Gostaria de dedicar este livro a minha mãe, por seu encorajamento durante o processo de criação do livro e por ser a melhor mãe que alguém poderia desejar, e a meu avô, Reginald Godfrey, por despertar em mim o interesse por ciência e eletrônica ainda quando jovem. Sem todos aqueles kits comprados na Radio Shack, nos Natais, eu talvez nunca tivesse chegado a escrever um livro sobre microcontroladores e eletrônica.

Obrigado a vocês dois.

Observação sobre figuras coloridas

No página do livro, em *novatec.com.br/catalogo/7522274_arduino/*, estão disponíveis para download versões coloridas de algumas figuras no livro.

Sumário

Sobre o autor

Michael McRoberts descobriu o Arduino em 2008, enquanto procurava formas de conectar um sensor de temperatura a um PC e criar um detector de nuvens para seu outro hobby, a astrofotografia. Depois de pesquisar um pouco, o Arduino pareceu-lhe a escolha óbvia, e o detector foi criado, com rapidez e sem custar caro. Iniciava assim o fascínio de Mike pelo Arduino. Desde então, ele realizou incontáveis projetos utilizando o Arduino. Também criou um kit para iniciantes e uma empresa online de comercialização de componentes, a Earthshine Electronics. Seu próximo projeto é utilizar um circuito com base em Arduino para enviar um balão de alta altitude ao limite da atmosfera, tirando fotos e gravando vídeos por diversão, com o auxílio do pessoal da UKHAS e da CUSF.

Seu interesse por eletrônica teve início ainda na infância, quando os kits de eletrônicos da Radio Shack preenchiam sua lista de presentes de Natal. Mike começou a programar como um hobby quando obteve um computador Sinclair ZX81, enquanto adolescente. Desde então, nunca ficou sem um computador. Recentemente tornou-se fã dos Macs.

Membro do London Hackspace e da Orpington Astronomical Society, Mike pode ser encontrado regularmente colaborando no Fórum do Arduino. Também gosta de participar de canais IRC sobre Arduino, altas altitudes e hacking (utilizando o nome "earthshine"), e do Twitter como "TheArduinoGuy". Quando não está trabalhando com Arduinos ou comandando a Earthshine Electronics, Mike gosta de dedicar seu tempo à astronomia, à astrofotografia, ao motociclismo e à navegação.

Sobre o revisor técnico

Josh Adams é desenvolvedor e arquiteto, com mais de nove anos de experiência profissional na criação de software de qualidade e em gerenciamento de projetos. Construiu uma bobina de Tesla para um projeto de ciência no segundo grau que emitia raios de quase 70 centímetros. Como arquiteto-chefe da Isotope Eleven, Josh é responsável por supervisionar decisões de arquitetura e por traduzir necessidades de clientes em software funcional. Josh formou-se na Universidade do Alabama em Birmingham (UAB) com bacharelado em ciências, tanto em Matemática quanto em Filosofia. Em seu tempo livre (que é muito raro), Josh realizou a revisão técnica para este livro sobre programação com o microprocessador Arduino. Quando não está trabalhando, Josh gosta de dedicar seu tempo a sua família.

Agradecimentos

Antes de tudo, gostaria de agradecer a minhas editoras Michelle Lowman e Jennifer Blackwell da Apress, uma vez que, sem elas, este livro nunca teria se tornado realidade; a meu revisor técnico, Josh Adams, por conferir pacientemente meu código e meus diagramas de circuito, deixando-os perfeitos; e a Nancy Wright, por encontrar todos os erros.

Um enorme agradecimento a todas as pessoas do Flickr e do Wikimedia Commons, que escolheram colocar suas imagens sob uma licença Creative Commons, e que me permitiram utilizá-las: Bruno Soares, Richard V. Gilbank, Inductiveload, Snorpey, Iain Fergusson, Patrick H. Lauke, cultured_society2nd, Cyril Buttay, Tony Jewell, Tod E. Kurt, Adam Grieg, David Stokes, Mike Prevette, David Mitchell, Aki Korhonen, Alex43223, Sparkfun, DFRobot, Adafruit Industries, Colin M.L. Burnett, David Batley, Jan-Piet Mens, Mercury13, Georg Wiora e Timo Arnall.

Obrigado a todos que me permitiram utilizar ou modificar seus códigos ou bibliotecas do Arduino para criar os projetos e que me ofereceram assistência técnica ou conselhos: Michael Margolis, Usman Haque from Pachube, Georg Kaindl, Tom Pollard, Jim Studt, Miles Burton, Robin James, Paul Stoffregen, Conor, Tom Igoe, Tim Newsome, James Whiddon, Bill Greiman, Matt Joyce, D. Sjunnesson, David A. Mellis, Bob S. (Xtalker), Ian Baker e NeoCat.

Agradeço à Sparkfun e à Adafruit Industries por me fornecerem as peças que utilizei e permitirem o uso de suas imagens. Agradeço também à equipe principal de desenvolvimento do Arduino, sem a qual o incrível Arduino e sua comunidade nem sequer existiriam: Massimo Banzi, Tom Igoe, David Cuartielles, Gianluca Martino, David Mellis e Nicholas Zambetti.

Por fim, obrigado a todas as pessoas do Fórum e do canal IRC do Arduino e também do Twitter, por sua ajuda, conselhos e encorajamento durante o processo deste livro, e ao London Hackspace, por me oferecer um local para testar alguns experimentos e escrever o capítulo final.

Caso tenha me esquecido de alguém, minhas desculpas e um agradecimento a vocês também.

Introdução

Descobri pela primeira vez o Arduino em 2008, quando procurava formas de conectar sensores de temperatura ao meu PC para que pudesse criar um detector de nuvens. Queria experimentar um conceito de detecção de nuvens a respeito do qual tinha lido em um fórum sobre clima e, como era experimental, não pretendia investir muito dinheiro nele, caso não desse certo. Havia muitas soluções disponíveis no mercado, mas o Arduino foi a que mais me agradou. Não apenas ele parecia uma forma fácil e de baixo custo para conectar os sensores de que eu necessitava, como também poderia ser utilizado para muitos outros propósitos. Milhares de projetos em blogs, sites com vídeos e fóruns mostravam projetos interessantes que as pessoas estavam realizando com seus Arduinos. Parecia haver um grande espírito de comunidade, com todos tentando ajudar uns aos outros.

Estava óbvio que eu poderia me divertir muito com o Arduino. Entretanto, eu não queria ter de pesquisar muitos sites em busca de informações. Preferia encontrar um livro sobre o assunto, algo que pudesse ter em minhas mãos e ler durante a viagem de trem para o trabalho. Depois de pesquisar, encontrei um livro. Infelizmente, ele era muito básico e estava desatualizado. Pior, não me oferecia nada prático para fazer com o Arduino, e também não gostei de seu estilo de didática. O que eu realmente desejava era um livro de abordagem prática que me ensinasse tanto programação quanto eletrônica enquanto criava os projetos, sem que, para isso, eu tivesse primeiro de ler páginas e páginas de teoria. Tal livro ainda não existia.

Então fundei a Earthshine Electronics para vender kits com base no Arduino. Para acompanhar o kit, produzi um livreto de tutorial que ajudaria as pessoas em seus primeiros passos. Esse livreto acabou se tornando muito popular, e recebi milhares de solicitações de pessoas perguntando quando adicionaria mais projetos, ou se venderia uma versão em separado. De fato, eu já havia pensado que seria ótimo produzir um livro completo para iniciantes, recheado de projetos e escrito em um estilo fácil de acompanhar. Foi dessa forma que este livro foi criado.

Escrevi este livro supondo que você nunca trabalhou com programação ou eletrônica. Também presumo que você não esteja interessado em ter de ler muita teoria antes de poder criar algo com o Arduino. Assim, desde o início do livro, você mergulhará diretamente na criação de um projeto simples. Depois, trabalhará em um total de 50 projetos, até que se torne confiante e capacitado para o desenvolvimento com o Arduino. Acredito que a melhor forma de aprender algo seja na prática e pondo a mão na massa.

O livro funciona da seguinte maneira: o primeiro projeto apresenta conceitos básicos sobre a programação do Arduino e também sobre eletrônica. O projeto seguinte constrói sobre esse conhecimento, indo um pouco mais além. Cada projeto, na sequência, acrescenta aos projetos anteriores. Ao concluir todos os 50 projetos, você estará confiante e capacitado para criar seus próprios projetos. Será capaz de adaptar suas novas habilidades e o conhecimento adquirido para conectar praticamente tudo ao seu Arduino e, dessa forma, realizar grandes projetos por diversão, ou para facilitar sua vida.

Cada projeto inicia com uma lista de componentes necessários. Escolhi componentes comuns que podem ser encontrados com facilidade. Também forneço um diagrama de circuito mostrando exatamente como conectar o Arduino e os componentes, utilizando fios jumper e uma protoboard (ou matriz de contato). Para criar as imagens dos componentes e os diagramas da protoboard para o livro, utilizei o excelente programa de código aberto Fritzing, que permite a projetistas documentarem seus protótipos e, depois, criarem layouts em placas de circuito impresso para fabricação. Trata-se de um excelente programa e uma forma brilhante de demonstrar para outras pessoas um circuito em uma protoboard. Não deixe de visitar *http://fritzing.org* para conferi-lo.

Depois de criado seu circuito, forneço uma listagem de código que você deve digitar no editor de programas do Arduino (o IDE), e depois fazer o upload para que seu projeto funcione. Você terá rapidamente um projeto completamente funcional. Somente depois de completado seu projeto, e de você tê-lo visto em funcionamento, é que explicarei como tudo funciona. O hardware será explicado de modo que você saiba como o componente funciona e como conectá-lo ao Arduino corretamente. O código será, então, explicado passo a passo para que você compreenda exatamente o que cada seção realiza. Ao dissecar o circuito e o código, você compreenderá como funciona o projeto como um todo e poderá, então, aplicar esse conhecimento e essas habilidades a projetos posteriores, e depois a seus próprios projetos no futuro.

O estilo de ensino que utilizo é muito fácil de acompanhar. Mesmo que você não tenha nenhuma experiência em programação ou eletrônica, será capaz de acompanhar o texto com facilidade e compreender os conceitos à medida que avançamos. Mais importante que tudo, você se divertirá. O Arduino é um produto excelente, divertido e de fonte aberta. Com o auxílio deste livro, você descobrirá como é fácil envolver-se na computação física e criar seus próprios dispositivos que interagem com o ambiente.

Mike McRoberts

CAPÍTULO 1

Introdução

Desde que o Arduino Project teve início em 2005, mais de 150.000 placas Arduino foram vendidas em todo o mundo. O número de placas-clone não oficiais sem dúvida supera o de placas oficiais, assim, é provável que mais de 500 mil placas Arduino e suas variantes tenham sido vendidas. Sua popularidade não para de crescer, e cada vez mais pessoas percebem o incrível potencial desse maravilhoso projeto de fonte aberta para criar projetos interessantes rápida e facilmente, com uma curva de aprendizagem relativamente pequena.

A maior vantagem do Arduino sobre outras plataformas de desenvolvimento de microcontroladores é a facilidade de sua utilização; pessoas que não são da área técnica podem, rapidamente, aprender o básico e criar seus próprios projetos em um intervalo de tempo relativamente curto. Artistas, mais especificamente, parecem considerá-lo a forma perfeita de criar obras de arte interativas rapidamente, e sem conhecimento especializado em eletrônica. Há uma grande comunidade de pessoas utilizando Arduinos, compartilhando seus códigos e diagramas de circuito para que outros os copiem e modifiquem. A maioria dessa comunidade também está muito disposta a auxiliar outros desenvolvedores. Você descobrirá que o Fórum do Arduino é o melhor local para buscar por respostas rápidas.

Entretanto, apesar da enorme quantidade de informação disponível aos iniciantes na Internet, a maioria desses dados está espalhada em várias fontes, fazendo com que seja complicado rastrear as informações necessárias. É aqui que este livro entra. Em suas páginas, temos 50 projetos criados para conduzi-lo passo a passo pela programação de seu Arduino. Quando você utiliza pela primeira vez seu Arduino (ou qualquer outro gadget, nesse sentido), deseja conectá-lo, ligar um LED e ver as luzes piscando imediatamente. Não quer ter de, primeiro, ler páginas e páginas de teoria. Como autor, compreendo a empolgação dos iniciantes, e justamente por isso você, desde o início, conectará dispositivos ao seu Arduino, fará o upload do código e verá tudo em funcionamento. Esta é, acredito, a melhor forma de aprender sobre um tema, especialmente um como a computação física, que é o tópico do Arduino.

Como utilizar este livro

O livro inicia com uma introdução sobre o Arduino, explicando como montar o hardware, instalar o software, fazer o upload de seu primeiro sketch e garantir que seu Arduino e o software estejam funcionando corretamente. Depois, explico o IDE do Arduino e como utilizá-lo, antes que você mergulhe em projetos que irão de criações básicas a tópicos avançados. Cada projeto inicia com uma descrição de como preparar o hardware, com o código necessário para que ele funcione. Depois, explico mais detalhadamente como o hardware e o código funcionam. Tudo será explicado em passos claros e simples, com muitos diagramas e fotos que tornarão muito fácil garantir que você acompanhe o projeto corretamente.

Você encontrará alguns termos e conceitos no livro que talvez não entenda a princípio. Não se preocupe, eles se tornarão claros à medida que você trabalhar nos projetos.

De que você necessita

Para ser capaz de acompanhar os projetos deste livro, você necessitará de diversos componentes. Como eles podem ser caros, sugiro que inicie comprando apenas os componentes para os projetos dos primeiros capítulos (os componentes listados no início das páginas dos projetos). Conforme avança pelo livro, você poderá adquirir os componentes necessários para os projetos subsequentes.

Há alguns outros itens de que você necessitará ou que pode considerar úteis. Evidentemente, você deverá obter uma placa Arduino ou uma das muitas placas-clone disponíveis no mercado, como Freeduino, Seeeduino (sim, é assim que se escreve), Boarduino, Sanguino, Roboduino ou qualquer uma das outras variantes "duino". Todas são compatíveis com o IDE do Arduino, com seus Shields e com tudo o mais que você pode utilizar com uma placa Arduino oficial. Lembre-se de que o Arduino é um projeto de código aberto e, portanto, livre para criação de clones ou de outras variantes. Entretanto, se você deseja apoiar a equipe de desenvolvimento da placa Arduino original, adquira uma placa oficial de um dos distribuidores reconhecidos. Em setembro de 2010, a mais recente variação da placa Arduino era a Arduino Uno.

Você deverá ter acesso à Internet para fazer o download do IDE (Integrated Development Environment, ou Ambiente de Desenvolvimento Integrado) do Arduino — o software utilizado para escrever seu código —; e também precisará da Internet para fazer o download dos exemplos de código deste livro (caso você não queira digitá-los manualmente) e de todas as bibliotecas de código que possam ser necessárias para que seu projeto funcione.

Você também necessitará de uma mesa bem iluminada, ou de outra superfície plana para dispor seus componentes; essa mesa deve estar próxima de seu PC ou laptop para permitir o upload do código para o Arduino. Lembre-se de que você está trabalhando com eletricidade (ainda que uma corrente contínua de baixa voltagem); portanto, uma superfície de metal deverá ser coberta por um material não condutivo, como uma toalha ou papel, antes que você posicione seus materiais. Da mesma forma, será interessante ter, ainda que não seja essencial, um par de alicates para cortar fios, alicates de ponta fina, e um desencapador de fios. Um caderno e uma caneta também serão úteis para esboçar diagramas aproximados, detalhar conceitos e projetos etc.

Por fim, os elementos necessários mais importantes são o entusiasmo e a disposição em aprender. O Arduino foi projetado como uma forma simples e barata de você se envolver com eletrônica de microcontroladores, e nada é difícil demais, desde que você esteja disposto a, ao menos, tentar. Este livro irá ajudá-lo nessa jornada, e apresentará a você esse empolgante e criativo hobby.

O que exatamente é um Arduino?

A Wikipédia afirma que "*Um Arduino é um microcontrolador de placa única e um conjunto de software para programá-lo. O hardware consiste em um projeto simples de hardware livre para o controlador, com um processador Atmel AVR e suporte embutido de entrada/saída. O software consiste de uma linguagem de programação padrão e do bootloader que roda na placa*."

Em termos práticos, um Arduino é um pequeno computador que você pode programar para processar entradas e saídas entre o dispositivo e os componentes externos conectados a ele (Figura 1.1). O Arduino é o que chamamos de plataforma de computação física ou embarcada, ou seja, um sistema que pode interagir com seu ambiente por meio de hardware e software. Por exemplo, um uso simples de um Arduino seria para acender uma luz por certo intervalo de tempo, digamos, 30 segundos, depois que um botão fosse pressionado. Nesse exemplo, o Arduino teria uma lâmpada e um botão conectados a ele. O Arduino aguardaria pacientemente até que o botão fosse pressionado; uma vez pressionado o botão, ele acenderia a lâmpada e iniciaria a contagem. Depois de contados 30 segundos, apagaria a lâmpada e aguardaria um novo apertar do botão. Você poderia utilizar essa configuração para controlar uma lâmpada em um closet, por exemplo.

Esse conceito poderia ser estendido pela conexão de um sensor, como um sensor de movimento PIR, para acender a lâmpada quando ele fosse disparado. Esses são alguns exemplos simples de como você poderia utilizar um Arduino.

O Arduino pode ser utilizado para desenvolver objetos interativos independentes, ou pode ser conectado a um computador, a uma rede, ou até mesmo à Internet para recuperar e enviar dados do Arduino e atuar sobre eles. Em outras palavras, ele pode enviar um conjunto de dados recebidos de alguns sensores para um site, dados estes que poderão, assim, ser exibidos na forma de um gráfico.

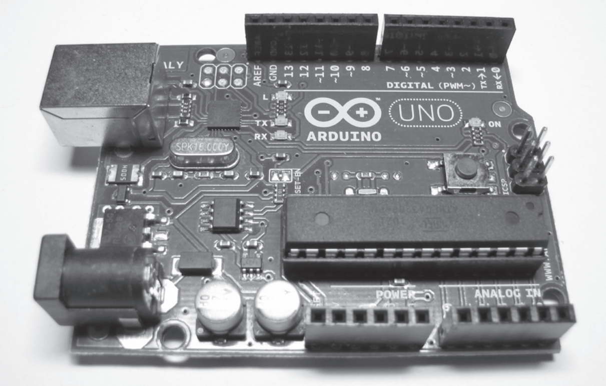


Figura 1.1 – Arduino Uno.

O Arduino pode ser conectado a LEDs, displays (mostradores) de matriz de pontos (Figura 1.2), botões, interruptores, motores, sensores de temperatura, sensores de pressão, sensores de distância, receptores GPS, módulos Ethernet ou qualquer outro dispositivo que emita dados ou possa ser controlado. Uma pesquisa na Internet revelará muitos outros projetos em que um Arduino foi utilizado para ler dados e controlar uma enorme quantidade de dispositivos.

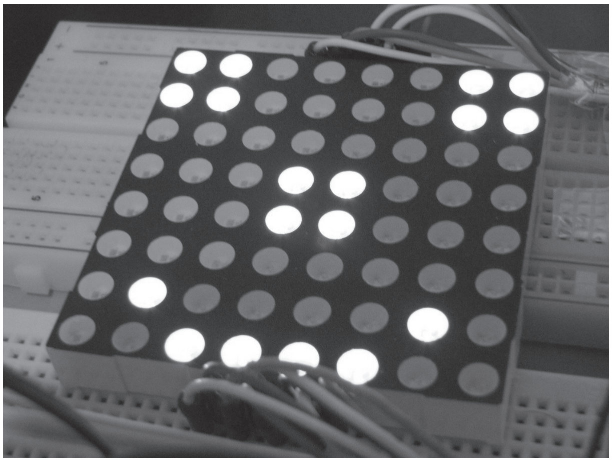


Figura 1.2 – Display de matriz de pontos controlado por um Arduino (imagem cortesia de Bruno Soares).

A placa do Arduino é composta de um microprocessador Atmel AVR , um cristal ou oscilador (relógio simples que envia pulsos de tempo em uma frequência especificada,

para permitir sua operação na velocidade correta) e um regulador linear de 5 volts. Dependendo do tipo de Arduino que você utiliza, ele também pode ter uma saída USB, que permite conectá-lo a um PC ou Mac para upload ou recuperação dos dados. A placa expõe os pinos de entrada/saída do microcontrolador, para que você possa conectá-los a outros circuitos ou sensores.

A mais recente placa do Arduino, a Uno, difere das versões prévias por não utilizar o chip FTDI, que conduz a USB para a serial. Em vez disso, ela utiliza um Atmega8U2, programado como um conversor USB para serial. Isso confere à placa muitas vantagens quando comparada à sua predecessora, a Duemilanove. Primeiro, o chip Atmega é muito mais barato que o chip FTDI, diminuindo o preço das placas. Segundo, e mais importante, ele permite que o chip USB tenha seu firmware atualizado, para que o Arduino seja exibido em seu PC como outro dispositivo, tal como um mouse ou joystick de jogos. Isso abre uma série de novas possibilidades para o Arduino. Infelizmente, a mudança para esse tipo de novo chip USB tornou muito mais difícil para fabricantes de clones criarem clones do Arduino Uno.

Para programar o Arduino (fazer com que ele faça o que você deseja) você utiliza o IDE do Arduino, um software livre no qual você escreve o código na linguagem que o Arduino compreende (baseada na linguagem C). O IDE permite que você escreva um *programa de computador*, que é um conjunto de instruções passo a passo, das quais você faz o upload para o Arduino. Seu Arduino, então, executará essas instruções, interagindo com o que estiver conectado a ele. No mundo do Arduino, programas são conhecidos como *sketches* (rascunho, ou esboço).

O hardware e o software do Arduino são ambos de fonte aberta, o que significa que o código, os esquemas, o projeto etc. podem ser utilizados livremente por qualquer pessoa e com qualquer propósito. Dessa forma, há muitas placas-clone e outras placas com base no Arduino disponíveis para compra, ou que podem ser criadas a partir de um diagrama. De fato, nada impede que você compre os componentes apropriados e crie seu próprio Arduino em uma matriz de pontos ou em sua PCB (Printed Circuit Board, placa de circuito impresso) feita em casa. A única ressalva que a equipe do Arduino impõe é que você não utilize a palavra "Arduino". Esse nome é reservado à placa oficial. Daí a existência de nomes para as placas-clone como Freeduino, Roboduino etc.

Como os projetos são de fonte aberta, qualquer placa-clone é 100% compatível com o Arduino e, dessa forma, qualquer software, hardware, shield etc. também será 100% compatível com o Arduino genuíno.

O Arduino também pode ser estendido utilizando os *shields* (escudos), que são placas de circuito contendo outros dispositivos (por exemplo, receptores GPS, displays de LCD, módulos de Ethernet etc.), que você pode simplesmente conectar ao seu Arduino para

obter funcionalidades adicionais. Os shields também estendem os pinos até o topo de suas próprias placas de circuito, para que você continue a ter acesso a todos eles. Você não tem de utilizar um shield se não quiser; pode fazer exatamente o mesmo circuito utilizando uma protoboard, Stripboard, Veroboard, ou criando sua própria PCB. A maioria dos projetos deste livro foi feita utilizando circuitos em uma protoboard.

Há muitas variantes diferentes do Arduino. A versão mais recente é o Arduino Uno. A versão anterior, a popular Duemilanove (2009 em italiano), é a placa que você provavelmente encontrará na vasta maioria dos projetos para Arduino na Internet. Você também pode obter versões Mini, Nano e Bluetooth do Arduino. Outra nova adição aos produtos disponíveis é o Arduino Mega 2560, que oferece mais memória e um número maior de pinos de entrada/saída. As novas placas utilizam um novo bootloader, o Optiboot, que libera mais 1,5 kB de memória flash e permite uma inicialização mais rápida.

Talvez o Arduino mais versátil, e daí o mais popular, seja o Arduino Uno (ou seu predecessor, o Duemilanove). Isso ocorre porque ele utiliza um chip padrão de 28 pinos, ligado a um soquete de circuito integrado (CI). A beleza desse sistema é que, se você criar alguma coisa com um Arduino e depois quiser transformá-la em algo permanente, em vez de utilizar uma placa Arduino relativamente cara, poderá simplesmente retirar o chip da placa e colocá-lo em sua própria placa de circuito, em seu dispositivo personalizado. Dessa forma, você tem um dispositivo embarcado personalizado, o que é muito bacana.

Então, com um pouco mais de investimento, você pode substituir o chip AVR em seu Arduino por um novo. Note que o chip deve ser pré-programado com o Arduino Bootloader (software programado no chip para habilitá-lo a utilizar o IDE do Arduino), mas você pode comprar um AVR Programmer para gravar o bootloader você mesmo, ou comprar um chip já programado; a maioria dos fornecedores de componentes para o Arduino oferece essas opções. Também é possível programar um chip utilizando um segundo Arduino; não é difícil encontrar instruções online nesse sentido.

Se você fizer uma pesquisa online por "Arduino", ficará surpreso com o grande número de sites dedicados ao Arduino, ou que apresentam projetos interessantes criados com ele. O Arduino é um dispositivo incrível, e possibilitará que você crie de tudo, desde obras de arte interativas (Figura 1.3) até robôs. Com um pouco de entusiasmo para aprender como programar um Arduino e fazê-lo interagir com outros componentes, assim como tendo um pouco de imaginação, você poderá construir tudo que quiser.

Este livro fornecerá as habilidades necessárias para que você dê seus primeiros passos nesse empolgante e criativo hobby. Agora que você já sabe o que é o Arduino, vamos ligá-lo ao seu computador e colocá-lo para funcionar.



Figura 1.3 – Instalação de arte Anthros, por Richard V. Gilbank, controlada utilizando um Arduino.

Primeiros passos

Esta seção explicará como configurar seu Arduino e o IDE pela primeira vez. São dadas instruções para Windows e Mac (executando o OSX 10.3.9 ou mais recente). Caso você esteja utilizando Linux, consulte as instruções Getting Started no site do Arduino, em *www.arduino.cc.playground/Learning/Linux*. Também suponho que você esteja utilizando um Arduino Uno. Se tiver uma placa diferente, como a Duemilanove (Figura 1.4), consulte a página correspondente no guia Getting Started no site do Arduino.

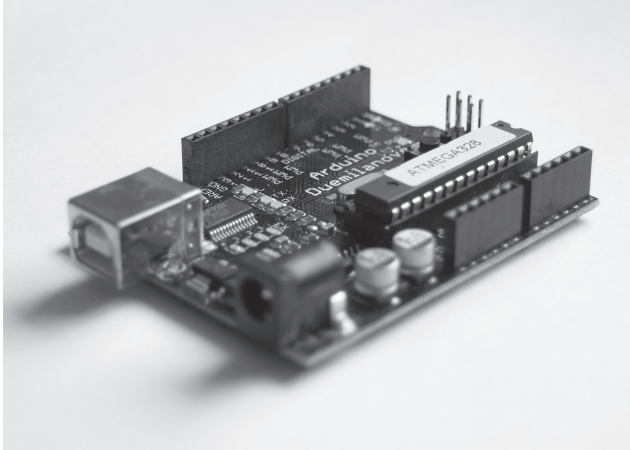


Figura 1.4 – Arduino Duemilanove (imagem cortesia de Snorpey).

Você também necessitará de um cabo USB (com plug do tipo A para B), do mesmo tipo utilizado na maioria das impressoras USB modernas. Se você tiver um Arduino Nano, necessitará de um cabo USB A para Mini-B. Não conecte ainda o Arduino, espere até que eu lhe diga para fazê-lo.

A seguir, faça o download do IDE do Arduino, o software que você utilizará para escrever seus programas (ou sketches) e, depois, fazer o upload deles para a placa. Para o IDE mais recente, acesse a página de download do Arduino, em *http://arduino.cc/en/Main/Software*, e obtenha a versão apropriada para seu sistema operacional.

Instalação no Windows XP

Assim que você tiver feito o download do IDE mais recente, descompacte o arquivo e clique duas vezes na pasta descompactada para abri-la. Você verá os arquivos do Arduino e suas subpastas. A seguir, conecte seu Arduino utilizando o cabo USB e verifique se o LED verde de energia (PWR) acende. O Windows lhe avisará "Novo hardware encontrado: Arduino" e o Assistente de instalação de novo hardware será exibido. Clique Avançar e o Windows tentará carregar os drivers. Esse processo falhará, mas não se preocupe, isso é normal.

Em seguida, clique com o botão direito no ícone Meu Computador em seu Desktop e escolha Gerenciar. A janela de Gerenciamento do Computador abrirá. Agora, desça até o Gerenciador de Dispositivos, na lista Ferramentas do Sistema, e clique nele. Na janela, à direita, você verá uma lista de seus dispositivos. O Arduino Uno fará parte dessa lista com um ponto de exclamação amarelo sobre ele, para indicar que o dispositivo não foi instalado corretamente. Clique com o botão direito sobre esse ícone e escolha "Atualizar Driver". Escolha "Não, não agora" na primeira página e clique em Avançar. Então, escolha "Instalar de uma lista ou local específico (avançado)" e clique novamente em Avançar. Agora clique em "Incluir este local na pesquisa" e clique em Procurar. Navegue até a pasta Drivers do IDE descompactado do Arduino e clique em Avançar. O Windows instalará o driver e você poderá, então, clicar no botão Concluir.

O Arduino Uno agora será exibido na opção Portas (COM & LPT), na lista de dispositivos, e mostrará o número de porta atribuído a ele (por exemplo, COM6). Para abrir o IDE, clique duas vezes no ícone do Arduino em sua pasta.

Instalação no Windows 7 e Vista

Assim que você tiver feito o download do IDE mais recente, descompacte o arquivo e clique duas vezes na pasta descompactada para abri-la. Você verá os arquivos do Arduino e suas subpastas. A seguir, conecte seu Arduino utilizando o cabo USB e verifique se o LED verde de energia (PWR) acende. O Windows tentará instalar automaticamente os drivers para o Arduino Uno e falhará. Isso é normal, por isso, não se preocupe.

Clique no botão Iniciar do Windows e depois em Painel de Controle. Agora, clique em Sistema e Segurança, depois em Sistema e, então, em Gerenciador de Dispositivos, na lista, no lado esquerdo. O Arduino fará parte da lista como um dispositivo com um ponto de exclamação amarelo sobre ele, para mostrar que não foi instalado corretamente. Clique com o botão direito no Arduino Uno e escolha Atualizar driver.

Depois, escolha Procurar software de driver no computador e, na janela seguinte, clique no botão Procurar. Navegue até a pasta *Drivers* dentro da pasta *Arduino* que você descompactou antes e clique em OK, e depois em Avançar. O Windows tentará instalar o driver. Uma caixa de segurança do Windows será exibida afirmando que o "Windows não pode verificar o editor deste software de driver". Clique em Instalar driver mesmo assim. A janela de instalação fará o necessário. *Se* tudo der certo, você terá outra janela dizendo que o software de driver foi atualizado com êxito. Finalmente, clique em Fechar. Para abrir o IDE, clique duas vezes no ícone do Arduino em sua pasta.

Instalação no Mac OSX

Faça o download do arquivo de imagem de disco (.*dmg*) mais recente para o IDE. Abra o arquivo .dmg (Figura 1.5).



Figura 1.5 – Arquivo .dmg do Arduino aberto no OSX.

Arraste o ícone do Arduino sobre a pasta Aplicações, soltando-o nela. Se você estiver utilizando um Arduino mais antigo, como o Duemilanove, terá de instalar o driver FTDI USB Serial. Clique duas vezes no ícone do pacote e siga as instruções para fazê-lo. Para o Uno e o Mega 2560, não será necessário instalar nenhum driver.

Para abrir o IDE, vá até a pasta Aplicações e clique no ícone do Arduino.

Seleção de placa e porta

Assim que você abrir o IDE, ele terá visual semelhante ao da figura 1.6.

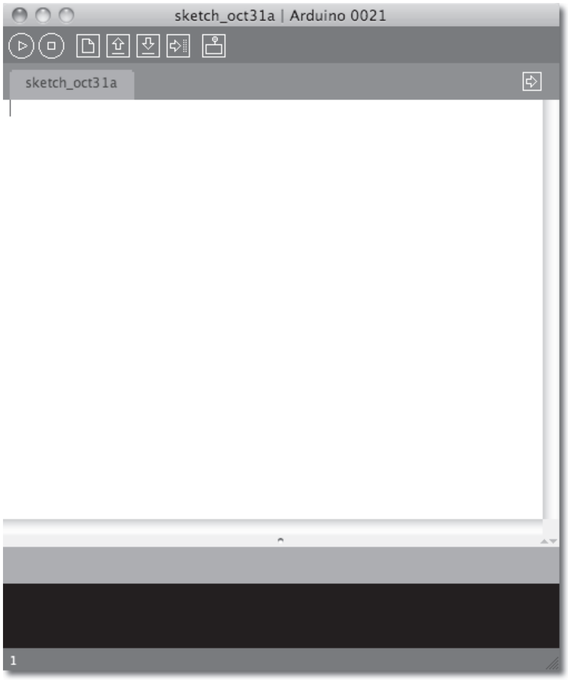


Figura 1.6 – IDE do Arduino quando aberto pela primeira vez.

Agora, vá ao menu e clique em Tools. Então, clique em Board (Figura 1.7).

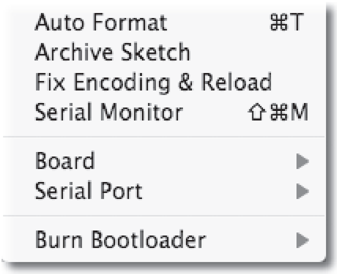


Figura 1.7 – Menu Tools do Arduino.

Você será apresentado a uma lista de placas (Figura 1.8). Se você tiver uma Uno, escolha de acordo. Se tiver uma Duemilanove, ou outra variante do Arduino, escolha a opção apropriada a partir dessa lista.

A seguir, clique no menu Tools novamente. Em seguida, clique em Serial Port e escolha na lista a porta apropriada para seu Arduino (Figura 1.9). Você agora está pronto para fazer o upload de um sketch de exemplo e testar se a instalação deu certo.



Figura 1.8 – Menu Boards do Arduino.



Figura 1.9 –Lista Serial Port.

Upload de seu primeiro sketch

Agora que você instalou os drivers e o IDE, e tem a placa e a porta correta selecionadas, é o momento de fazer o upload de um sketch de exemplo para o Arduino e testar se tudo está funcionando corretamente, antes de avançar ao seu primeiro projeto.

Primeiro, clique no menu File (Figura 1.10) e depois em Examples.

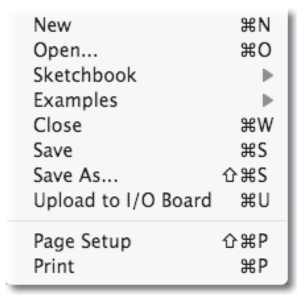


Figura 1.10 – Menu File.

Você será apresentado a uma enorme lista de exemplos. Vamos experimentar um bem simples. Clique em Basics e depois em Blink (Figura 1.11). O sketch Blink será carregado no IDE.

Depois, clique no botão Upload (o sexto botão a partir da esquerda) e olhe para seu Arduino. (Se você tem um Arduino Mini, NG ou outra placa, pode ser necessário apertar o botão reset na placa antes de apertar o botão Upload). As luzes RX e TX devem começar a piscar para mostrar que os dados estão sendo transmitidos de seu

computador para a placa. Assim que o upload do sketch tiver sido feito, as palavras "Done uploading" (upload completo) serão exibidas na barra de status do IDE e as luzes RX e TX pararão de piscar.

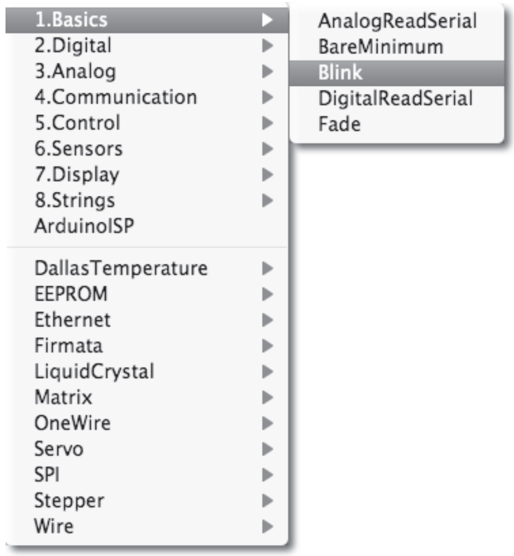


Figura 1.11 – Menu Examples.

Depois de alguns segundos, você deverá ver que o LED do pino 13 (o pequenino LED ao lado dos LEDs RX e TX) começa a piscar, acendendo e apagando em intervalos de um segundo. Se isso estiver acontecendo, você conectou seu Arduino, instalou os drivers e o software, e fez o upload de um sketch de exemplo com êxito. O sketch Blink é um sketch muito simples que pisca o LED 13 (Figura 1.12), o pequenino LED verde (ou laranja) soldado na placa (e que também está conectado ao pino digital 13 do microcontrolador).

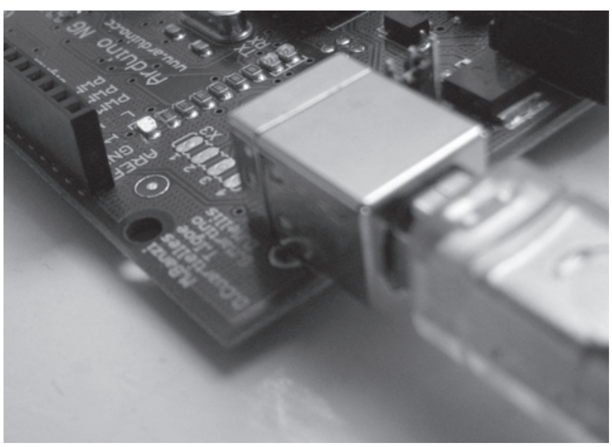


Figura 1.12 – LED 13 piscando.

Antes de avançar ao Projeto 1, vamos analisar o IDE do Arduino. Explicarei cada parte do programa.

IDE do Arduino

Quando você abrir o IDE do Arduino, verá algo semelhante à figura 1.13. Caso esteja utilizando Windows ou Linux, pode haver algumas pequenas diferenças, mas o IDE é basicamente o mesmo, independentemente do sistema operacional utilizado.

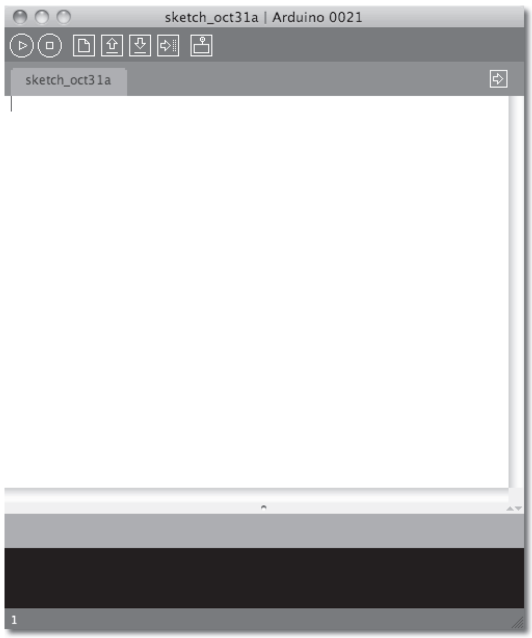


Figura 1.13 – Visual do IDE quando a aplicação abre.

O IDE é dividido em três partes: a Toolbar no topo, o código ou a Sketch Window no centro, e a janela de mensagens na base. A Toolbar consiste de sete botões. Sob a Toolbar há uma guia, ou um conjunto de guias, com o nome do arquivo do sketch. Também há um botão posicionado no lado direito.

Ao longo do topo há a barra de menus, com os itens File, Edit, Sketch, Tools e Help. Os botões na Toolbar (Figura 1.14) fornecem acesso conveniente às funções mais utilizadas dentro desses menus.

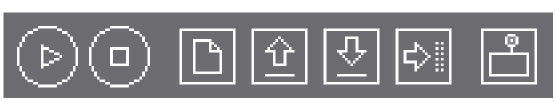


Figura 1.14 – Toolbar.

Os botões da Toolbar e suas funções estão listados na tabela 1.1.

Tabela 1.1 – Funções dos botões da Toolbar

Verify/Compile	Verifica se há erros no código
Stop	Interrompe o monitor serial, ou desmarca os outros botões
New	Cria um sketch em branco
Open	Mostra uma lista de sketches, em seu Sketchbook, para abrir
Save	Salva o sketch atual em seu sketchbook
Upload	Faz o upload do sketch atual para o Arduino
Serial Monitor	Exibe os dados seriais enviados do Arduino

O botão Verify/Compile é utilizado para verificar se seu código está correto e livre de erros, antes que você faça o upload para a placa do Arduino.

O botão Stop interrompe a operação do monitor serial e também desmarca outros botões selecionados. Enquanto o monitor serial está em operação, você pode pressionar o botão Stop para obter um instantâneo dos dados seriais, até então enviados, e analisá-los. Isso é especialmente útil se você estiver enviando dados para o monitor serial mais rápido do que puder lê-los.

O botão New cria um novo sketch em branco, pronto para receber seu código. O IDE pede que você digite um nome e forneça uma localização para seu sketch (tente utilizar a localização padrão, se possível) e, então, oferece um sketch em branco, pronto para receber seu código. A guia no topo do sketch exibe o nome que você deu a ele.

O botão Open apresenta uma lista de sketches armazenados em seu sketchbook, bem como uma lista de sketches de exemplo que você pode experimentar com diversos periféricos. Os sketches de exemplo são muito valiosos para iniciantes, por formarem uma base a partir da qual poderão construir seus próprios sketches. Abra o sketch apropriado para o dispositivo com o qual está conectando, e modifique o código conforme sua necessidade.

O botão Save salva o código na janela de sketch para seu arquivo. Uma vez feito isso, você receberá uma mensagem “Done saving” (“Gravação completa”) na base de sua janela de código, indicando que o sketch foi salvo.

O botão Upload to I/O Board faz o upload do código contido na janela de sketch atual para seu Arduino. Certifique-se de que você selecionou corretamente a placa e a porta (no menu Tools) antes do upload. É essencial que você salve seu sketch antes de fazer o upload para sua placa, caso ocorra algum erro que trave seu sistema ou o IDE. Também é aconselhável pressionar o botão Verify/Compile antes do upload para garantir que não existam erros que terão de ser depurados primeiro.

O monitor serial é uma ferramenta muito útil, especialmente para depuração de código. O monitor exibe os dados seriais enviados de seu Arduino (USB ou placa serial). Você também pode enviar dados de volta ao Arduino utilizando o monitor serial. Clique no botão **Serial Monitor** para abrir uma janela como a da figura 1.15.

No canto inferior direito você pode ver a taxa de transmissão (Baud Rate) na qual os dados seriais devem ser enviados de/para o Arduino. A taxa de transmissão é a taxa por segundo em que alterações de estado ou bits (dados) são enviados de/para a placa. A configuração padrão é 9.600 baud, o que significa que, se você quisesse enviar um livro pela linha de comunicação serial (nesse caso, seu cabo USB), 1.200 letras ou símbolos de texto seriam enviados por segundo (9.600 bits / 8 bits por caractere = 1.200 bytes ou caracteres). Bits e bytes serão explicados futuramente.

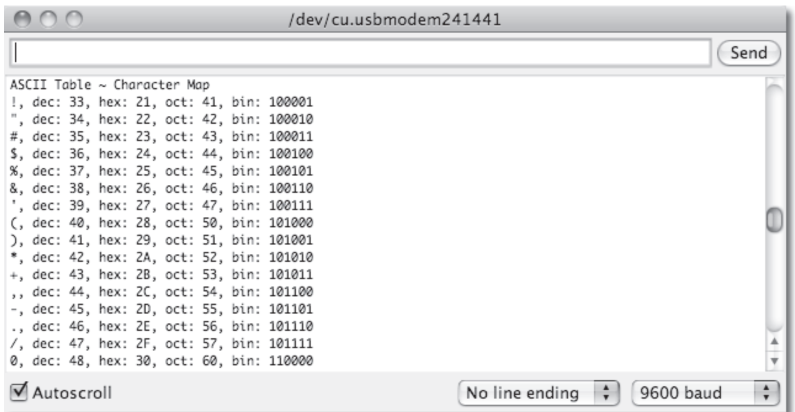


Figura 1.15 – Janela serial em uso.

No topo, há uma caixa de texto em branco, para que você digite o texto a ser enviado de volta para o Arduino, e um botão Send, para enviar o texto. Note que o monitor serial não pode receber nenhum dado serial, a menos que você tenha preparado o código em seu sketch para que isso ocorra. Da mesma forma, o Arduino não receberá nenhum dado, a menos que você o tenha codificado para tanto.

Por fim, a área central da janela é o local em que seus dados seriais serão exibidos. Na imagem anterior, o Arduino está executando o sketch ASCIITable (do exemplo Communications). Esse programa faz a saída de caracteres ASCII do Arduino pela serial (o cabo USB) para o PC, onde o monitor serial os exibe.

Para iniciar o monitor serial, pressione o botão **Serial Monitor**. Para interrompê-lo, pressione o botão **Stop**. Em um Mac ou Linux, a placa do Arduino reiniciará sozinha (reexecutando o código desde o início), quando você clicar no botão **Serial Monitor**.

Assim que você estiver capacitado para a comunicação serial de, e para, o Arduino, poderá utilizar outros programas, como Processing, Flash, MaxMSP etc. para realizar a comunicação entre o Arduino e seu PC. Você utilizará o monitor serial futuramente, ao ler dados de sensores e fazer com que o Arduino envie esses dados para o monitor serial em um formato legível para seres humanos.

Na parte inferior da janela do IDE você verá mensagens de erro (em texto vermelho) que o IDE poderá exibir quando tentar se conectar com sua placa, realizar o upload do código ou verificá-lo. No canto inferior esquerdo do IDE você verá um número. Essa é a localização atual do cursor dentro do programa. Se você tiver código em sua janela, e mover o cursor para baixo (utilizando a tecla ↓ de seu teclado), verá que esse número se eleva. Isso é útil para encontrar bugs destacados por mensagens de erro.

No topo da janela do IDE (ou no topo de sua tela, se você estiver utilizando um Mac), você verá os diversos menus em que pode clicar para acessar mais itens de menu (Figura 1.16).



Figura 1.16 – Menus do IDE.

O primeiro menu é o menu Arduino (Figura 1.17). A opção About Arduino mostra o número da versão atual, uma lista das pessoas envolvidas na criação desse fantástico dispositivo e algumas informações adicionais.



Figura 1.17 – Menu Arduino.

Sob essa primeira opção, há a opção Preferences, que exibe a janela de preferências em que você pode alterar várias opções do IDE, como a localização padrão de seu sketchbook etc. A opção Quit encerra o programa.

O menu File (Figura 1.18) é o local em que você pode acessar opções para criar um novo sketch, analisar sketches armazenados em seu sketchbook (bem como sketches de exemplo), salvar seu sketch ou utilizar a opção Save As para dar a ele um novo nome, fazer o upload de seu sketch para a placa de entrada/saída (o Arduino), ou imprimir seu código.

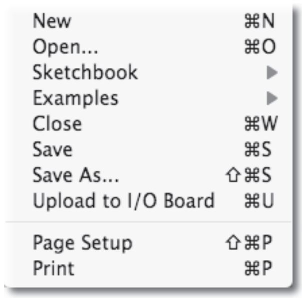


Figura 1.18 – Menu File.

O menu Edit (Figura 1.19) oferece opções para que você recorte, copie e cole seções do código. Você também pode selecionar todo o código (Select All) ou localizar (Find) certas palavras ou frases dentro do código. As opções Undo (desfazer) ou Redo (refazer) são úteis quando você comete algum erro.

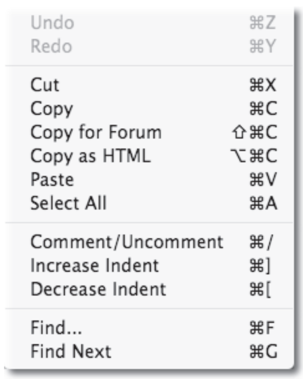


Figura 1.19 – Menu Edit.

O menu Sketch (Figura 1.20) contém as funções Verify/Compile e outras funções úteis, incluindo a opção Import Library, que apresenta uma lista das bibliotecas disponíveis armazenadas em sua pasta libraries.

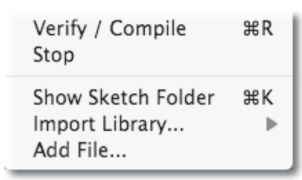


Figura 1.20 – Menu Sketch.

Uma *biblioteca* é um conjunto de código que você pode incluir em seu sketch para aprimorar a funcionalidade de seu projeto. Isso é uma forma de impedir que você tenha de reinventar a roda, recriando algo que já foi feito; em vez disso, você pode reutilizar código escrito por outra pessoa para diversos componentes de hardware comuns. Por exemplo, a biblioteca Stepper é um conjunto de funções para controlar

um motor de passo; assim, incluindo essa biblioteca em seu sketch, você pode reutilizar o código diversas vezes em projetos diferentes. Dessa forma, você também pode ocultar do usuário as partes mais complicadas de seu código. Futuramente, falarei com mais detalhes sobre o uso de bibliotecas.

A opção Show Sketch Folder abre a pasta em que o sketch está armazenado. A opção Add File permite que você adicione outro arquivo de fonte ao seu sketch, permitindo a divisão de sketches maiores em arquivos menores para, depois, adicioná-los ao sketch principal.

O menu Tools (Figura 1.21) oferece diversas opções. Você pode selecionar a placa e a porta serial como fez quando configurou o Arduino pela primeira vez. A função Auto Format formata seu código para melhorar sua visualização. A opção Copy for Forum copia o código da janela de sketch, mas em um formato que, quando colado no fórum do Arduino (e na maioria dos outros fóruns), seja exibido da mesma forma que no IDE, acompanhado da colorização da sintaxe etc. A opção Archive Sketch permite que você comprima seu sketch em um arquivo ZIP, perguntando onde você deseja armazená-lo. Por fim, a opção Burn Bootloader grava o Arduino Bootloader (o trecho de código do chip para torná-lo compatível com o IDE do Arduino) no chip. Essa opção pode apenas ser utilizada se você tiver um programador AVR e se tiver substituído o chip em seu Arduino, ou caso tenha comprado chips em branco para utilizar em seu próprio projeto embarcado. A menos que pretenda gravar muitos chips, geralmente é mais barato simplesmente comprar um chip ATmega (Figura 1.22) com o Arduino Bootloader pré-programado. Muitas lojas online oferecem chips pré-programados a ótimos preços.

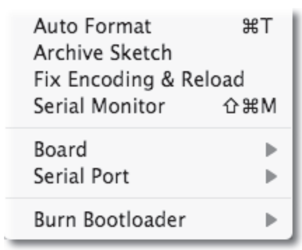


Figura 1.21 – Menu Tools.

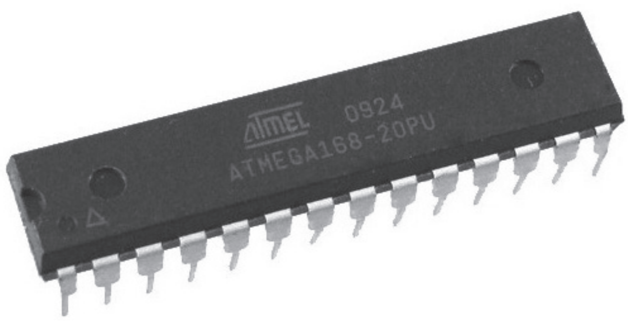


Figura 1.22 – Chip Atmel ATmega, o coração de seu Arduino (imagem cortesia da Earthshine Electronics).

O último menu, **Help**, é o local em que você encontra mais informações sobre o IDE, ou links para as páginas de referência do site do Arduino e outras páginas úteis.

O IDE do Arduino é bem simples, e você aprenderá como utilizá-lo com rapidez e facilidade à medida que avança nos projetos. Conforme você se torna mais proficiente no uso do Arduino e na programação em C (a linguagem de programação utilizada para criar código no Arduino é um dialeto da linguagem C), poderá acabar considerando o IDE do Arduino como sendo básico demais. Caso queira algo com mais funcionalidades, poderá experimentar algum dos programas de IDE profissionais (mesmo os gratuitos), como Eclipse, ArduIDE, GNU/Emacs, AVR-GCC, AVR Studio e até mesmo o XCode da Apple.

Agora que você tem o software de seu Arduino instalado, a placa conectada e funcionando, e um conhecimento básico de como utilizar o IDE, vamos avançar diretamente para o Projeto 1, no qual criaremos um LED piscante.

CAPÍTULO 2

Acendendo as luzes

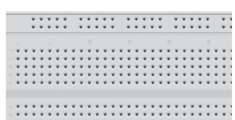
Neste capítulo, você trabalhará em seus quatro primeiros projetos. Todos utilizam luzes LED de diversas formas. Você aprenderá a controlar as saídas do Arduino e também entradas simples, como botões pressionados. No lado do hardware, você aprenderá sobre LEDs, botões e resistores, incluindo resistores pull-up e pull-down, importantes para garantir que os dispositivos de entrada sejam lidos corretamente. Nesse processo, você aprenderá conceitos de programação com a linguagem do Arduino. Vamos começar com um projeto "Olá mundo", que faz seu Arduino piscar um LED externo.

Projeto 1 – LED piscante

Para o primeiro projeto, você repetirá o sketch que utilizou no estágio de testes. Dessa vez, no entanto, você conectará um LED a um dos pinos digitais em vez de utilizar o LED 13, soldado na placa. Você também aprenderá exatamente como o hardware e o software para este projeto funcionam, aprendendo, ao mesmo tempo, um pouco sobre eletrônica e codificação na linguagem do Arduino (que é uma variante da linguagem C).

Componentes necessários

Protoboard



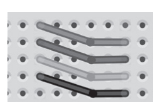
LED de 5 mm



Resistor de 100 ohms*



Fios jumper



* Esse valor pode ser diferente, dependendo do LED que você utilizar. O texto explicará como descobrir o valor correto.

O melhor tipo de protoboard para a maioria dos projetos deste livro é uma protoboard de 840 pontos. Protoboards desse tipo costumam ser de tamanho padrão, medindo aproximadamente 16,5 cm por 5,5 cm e apresentando 840 furos (ou pontos) na placa. Geralmente, as placas têm pequenos encaixes nas laterais que permitem conectar diversas placas, umas às outras, para criar protoboards maiores; isso é útil para projetos mais complexos. Para este projeto, entretanto, uma protoboard de tamanho normal será suficiente.

O LED deverá ser de 5 mm, de uma cor qualquer. Você também deverá saber qual a corrente e a voltagem (por vezes chamadas de corrente contínua e voltagem contínua) do LED, para poder calcular o valor do resistor necessário — isso será feito mais adiante no projeto.

Os fios jumper que utilizaremos podem ser fios encontrados comercialmente (geralmente com pontas moldadas para facilitar sua inserção na protoboard), ou você pode criar seus próprios, cortando tiras curtas de fios rígidos de núcleo único e retirando cerca de 6 mm da ponta.

Conectando os componentes

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue sua protoboard, o LED, o resistor e os fios, e conecte tudo como mostra a figura 2.1.

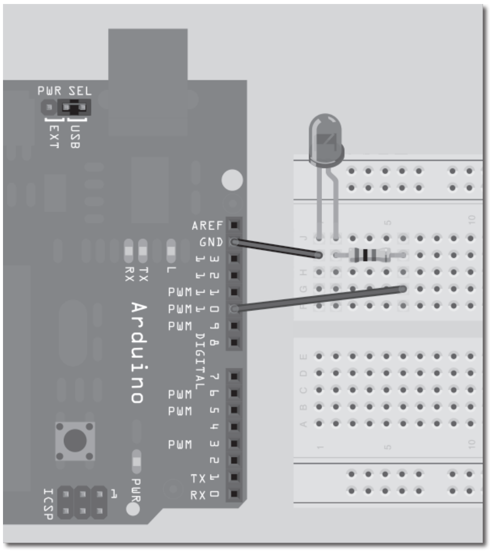


Figura 2.1 – Circuito para o Projeto 1 – LED piscante (consulte o site da Novatec para versão colorida).

Não importa se você utiliza fios de cores diferentes ou furos diferentes na protoboard, desde que os componentes e os fios estejam conectados na mesma ordem da figura.

Tenha cuidado ao inserir os componentes na protoboard. Caso sua protoboard seja nova, a superfície dos furos ainda estará rígida. A não inserção cuidadosa dos componentes pode resultar em danos.

Certifique-se de que seu LED esteja conectado corretamente, com o terminal (ou perna) mais longo conectado ao pino digital 10. O terminal longo é o ânodo do LED, e deve sempre ir para a alimentação de +5 V (nesse caso, saindo do pino digital 10); o terminal curto é o cátodo e deve ir para o terra (GND).

Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

Digite o código

Abra seu IDE do Arduino e digite o código da listagem 2.1.

Listagem 2.1 – Código para o projeto 1

```
// Projeto 1 - LED piscante
int ledPin = 10;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Pressione o botão **Verify/Compile** no topo do IDE para certificar-se de que não há erros em seu código. Se não houver erros, clique no botão Upload para fazer o upload do código ao seu Arduino. Caso tudo tenha sido feito corretamente, agora você deverá ver o LED vermelho, na protoboard, acendendo e apagando em intervalos de um segundo.

Vamos analisar o código e o hardware para descobrir como ambos funcionam.

Projeto 1 – LED piscante – Análise do código

A primeira linha do código do projeto é:

```
// Projeto 1 - LED piscante
```

Trata-se apenas de um *comentário* em seu código. Você pode perceber que é um comentário porque ela inicia com `//`, e qualquer texto que inicie dessa forma é ignorado pelo compilador. Comentários são essenciais em seu código; eles ajudam a compreender como seu código funciona. À medida que seus projetos se tornarem mais complexos, e seu código se expandir até centenas ou talvez milhares de linhas, comentários

serão vitais para facilitar a compreensão de como cada seção funciona. Você pode desenvolver um trecho incrível de código, mas não pode simplesmente supor que se lembrará de como ele funciona quando revisitá-lo muitos dias, semanas ou meses depois. Comentários, por outro lado, poderão ajudá-lo a recordar a funcionalidade do código. Da mesma forma, caso seu código seja visto por outras pessoas, comentários poderão ajudar esses indivíduos a compreender o que ocorre nele. O espírito do Arduino, e de toda a comunidade de fonte aberta, trata do compartilhamento de código e projetos. Espero que, quando você vier a criar seus próprios projetos com o Arduino, também esteja disposto a compartilhá-los com o mundo.

Há outro formato para comentários: um bloco de instrução dentro dos sinais `/*` e `*/`, da seguinte maneira:

```
/* Todo o texto dentro
das barras e dos asteriscos
é um comentário, e será
ignorado pelo compilador */
```

O IDE transformará automaticamente a cor de todos os comentários em cinza. A linha seguinte no programa é

```
int ledPin = 10;
```

Isso é que chamamos de *variável*. Uma variável é um local em que podemos armazenar dados. Nesse caso, você está definindo uma variável de tipo `int`, ou inteiro. Um *inteiro* é um número dentro do intervalo de -32.768 e 32.767. Em seguida, você atribui a esse inteiro o nome `ledPin` e dá a ele um valor de `10`. (Você não tinha de chamá-lo `ledPin`, poderia ter dado o nome que quisesse. Mas você deve sempre procurar fazer com que suas variáveis tenham nomes descritivos, por isso `ledPin`, para mostrar que esta variável define qual pino no Arduino você utilizará para conectar o LED.) Nesse caso você está utilizando o pino digital 10. Ao final da instrução há um ponto e vírgula. Esse símbolo diz ao compilador que a instrução, agora, está completa.

Ainda que você possa dar qualquer nome às suas variáveis, todas as variáveis em C devem iniciar com uma letra; o restante do nome pode ser formado por letras, números e underscores. Note que a linguagem C diferencia caracteres maiúsculos e minúsculos. Por fim, você não pode utilizar nenhuma das palavras-chave da linguagem, como `main`, `while`, `switch` etc. como nomes de variáveis. Palavras-chave são nomes de constantes: variáveis e funções definidas como parte da linguagem do Arduino. Para evitar que você nomeie uma variável como uma palavra-chave, todas as palavras-chave no sketch serão mostradas em vermelho.

Imagine uma variável como uma pequena caixa, na qual você pode guardar objetos. Assim, nesse sketch, você preparou uma área na memória para armazenar um número de valor inteiro, e armazenou nessa área o número 10.

Por fim, uma variável é chamada de "variável" porque você pode modificá-la. Futuramente, você realizará cálculos matemáticos em variáveis, para fazer com que seu programa realize funções mais avançadas.

Em seguida, vemos a função `setup()`:

```
void setup() {
  pinMode(ledPin, OUTPUT);
}
```

Um sketch do Arduino deve ter uma função `setup()` e uma função `loop()`, do contrário, não funcionará. A função `setup()` é executada somente uma vez no início do programa, e é nela que você emitirá instruções gerais para preparar o programa antes que o loop principal seja executado, como a definição dos modos dos pinos, das taxas de transmissão serial etc. Basicamente, uma função é uma porção de código agrupada em um bloco conveniente. Por exemplo, se você tivesse criado sua própria função para realizar uma série de complicadas operações matemáticas, e que tivesse muitas linhas de código, poderia executar esse código quantas vezes quisesse simplesmente chamando o nome da função, em vez de ter de reescrever o código cada vez que fosse usá-lo. Você verá funções em maiores detalhes no futuro, quando começar a criar suas próprias. No caso desse programa, todavia, a função `setup()` tem somente uma instrução para executar. A função inicia com

```
void setup()
```

Isso diz ao compilador que sua função é chamada `setup`, que ela não retorna nenhum dado (`void`) e que você não passa nenhum parâmetro a ela (parênteses vazios). Se sua função retornasse um valor inteiro e você também tivesse valores inteiros a serem passados a ela (por exemplo, para serem processados pela função), o resultado seria algo como:

```
int myFunc(int x, int y)
```

Aqui você criou uma função (ou um bloco de código) de nome `myFunc`. A ela foram passados dois inteiros, `x` e `y`. Assim que tiver concluído, ela retornará um valor inteiro, no ponto de execução imediatamente depois de onde havia sido chamada no programa (portanto, utilizamos `int` antes do nome da função).

Todo o código dentro da função está contido entre chaves. Um símbolo `{` inicia o bloco de código, e um símbolo `}` termina o bloco. Tudo que existir entre esses dois símbolos, no código, fará parte da função. (Falarei mais detalhadamente sobre funções no futuro, por isso não se preocupe com elas, por enquanto.)

Nesse programa, você tem duas funções; a primeira é chamada `setup`, e seu propósito é preparar tudo que é necessário para que seu programa funcione antes da execução do loop principal do programa:

```
void setup() {
  pinMode(ledPin, OUTPUT);
}
```

Sua função `setup` tem apenas uma instrução, `pinMode`, que diz ao Arduino que você deseja definir o modo de um de seus pinos como Saída (Output), e não Entrada (Input). Dentro dos parênteses, você coloca o número do pino e o modo (`OUTPUT` ou `INPUT`). O número de seu pino é `ledPin`, previamente definido com o valor `10`. Dessa forma, essa instrução está simplesmente dizendo ao Arduino que o pino digital 10 deve ser definido como modo `OUTPUT`. Como a função `setup()` executa apenas uma vez, agora você avança para o loop principal da função:

```
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

A função `loop()` é a função principal do programa e executa continuamente enquanto o Arduino estiver ligado. Todas as declarações dentro da função `loop()` (dentro das chaves) são executadas uma de cada vez, passo a passo, até que se alcance o fim da função; nesse ponto, o loop reinicia desde o princípio e assim infinitamente, ou até que o Arduino seja desligado ou o botão Reset pressionado.

Neste projeto, você deseja que o LED acenda, fique aceso por um segundo, apague, permaneça apagado por um segundo e então repita o processo. Os comandos para dizer ao Arduino como fazer essas operações estão dentro da função `loop()`, pois você deseja que sejam repetidos seguidas vezes. A primeira instrução é:

```
digitalWrite(ledPin, HIGH);
```

Ela escreve um valor `HIGH` ou `LOW` para o pino dentro da instrução (nesse caso `ledPin`, que é o pino digital 10). Quando você define um pino como `HIGH`, está enviando 5 volts para ele. Quando define como `LOW`, o pino se torna 0 volt, ou terra. Essa instrução, portanto, envia 5 V para o pino 10 e acende o LED. Depois dela, temos:

```
delay(1000);
```

Essa instrução simplesmente diz ao Arduino para esperar 1.000 milissegundos (há 1.000 milissegundos em um segundo) antes de executar a instrução seguinte:

```
digitalWrite(ledPin, LOW);
```

O que desliga a força que vai para o pino digital 10 e apaga o LED. Então, há outra instrução de espera por mais 1.000 milissegundos, e depois a função termina. Entretanto, como essa é sua função `loop()` principal, a função reiniciará desde o princípio.

Seguindo a estrutura do programa passo a passo novamente, você pode ver que tudo é muito simples:

```
// Projeto 1 - LED piscante
int ledPin = 10;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Você inicia atribuindo uma variável `ledPin`, e dando a ela um valor de `10`. Depois, avança para a função `setup()`, na qual você define o modo para o pino digital 10 como saída. No loop principal do programa, você define o pino digital 10 como `HIGH`, enviando a ele 5 V. Então, espera por um segundo e desliga os 5 V, antes de esperar mais um segundo. O loop então reinicia desde o princípio: o LED acenderá e apagará continuamente, enquanto o Arduino tiver energia.

Agora que você sabe de tudo isso, pode modificar o código para acender o LED por um intervalo diferente de tempo e desligá-lo por outro intervalo. Por exemplo, se você quisesse que o LED ficasse aceso por dois segundos e, depois, apagado por meio segundo, poderia fazer o seguinte:

```
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(2000);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

Se quisesse que o LED ficasse apagado por cinco segundos e, depois, piscasse brevemente (250ms), como o indicador LED de um alarme de carro, poderia fazer o seguinte:

```
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(250);
  digitalWrite(ledPin, LOW);
  delay(5000);
}
```

Para que o LED pisque, acendendo a apagando rapidamente, tente:

```
void loop() {
  digitalWrite(ledPin, HIGH);
```

```
    delay(50);
    digitalWrite(ledPin, LOW);
    delay(50);
  }
```

Alternando o tempo em que o LED fica aceso e apagado, você pode criar o efeito que quiser (dentro do que pode ser feito com um único LED). Antes de avançarmos para algo mais empolgante, vamos analisar o hardware e ver como ele funciona.

Projeto 1 – LED piscante – Análise do hardware

O hardware utilizado no projeto 1:

Protoboard



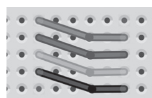
LED de 5 mm



Resistor de 100 ohms*



Fios jumper



A protoboard é um dispositivo reutilizável, sem solda, utilizado para prototipar um circuito eletrônico ou para experimentar projetos de circuitos. A placa consiste em uma série de furos em uma grade; sob a placa, esses furos são conectados por uma tira de metal condutivo. A forma como essas tiras são dispostas é tipicamente a que vemos na figura 2.2.

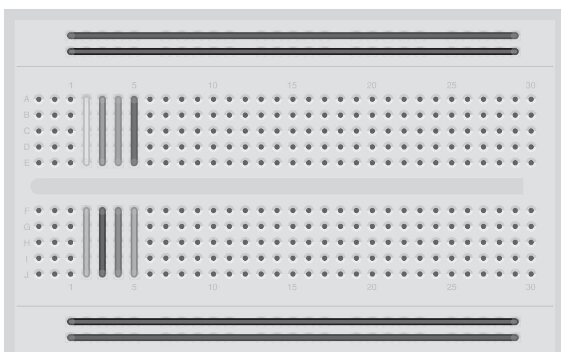


Figura 2.2 – Como são dispostas as tiras de metal em uma protoboard.

As tiras ao longo do topo e da base correm em paralelo à placa, e são projetadas para carregar o barramento de alimentação e o barramento do terra. Os componentes no meio da placa convenientemente conectam com os 5 V (ou a voltagem que você estiver

utilizando) ou com o terra. Algumas protoboards têm uma linha vermelha e outra preta correndo paralelas a esses furos, para mostrar qual é a alimentação (vermelho) e qual é o terra (preto). Em protoboards maiores, o barramento de alimentação às vezes tem uma divisão, indicada por uma quebra na linha vermelha. Isso torna possível enviar voltagens diferentes para partes distintas de sua placa. Caso você esteja utilizando apenas uma voltagem, um pequeno pedaço de fio jumper pode ser colocado de um lado a outro desse espaço, para garantir que a mesma voltagem seja aplicada em todo o barramento.

As tiras no centro correm a 90 graus dos barramentos de alimentação e do terra, e há um espaço vazio no meio para que você possa colocar Circuitos Integrados, de modo que cada pino do chip vá para um conjunto diferente de furos e, portanto, para um barramento diferente (Figura 2.3).

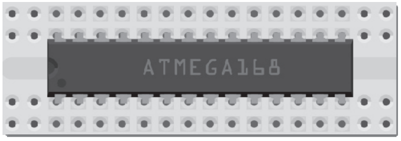


Figura 2.3 – Circuito Integrado (ou chip) conectado no espaço de uma protoboard.

O próximo componente é um *resistor*, um dispositivo projetado para provocar resistência a uma corrente elétrica, causando uma queda na voltagem em seus terminais. Você pode pensar em um resistor como um cano de água muito mais fino do que o cano conectado a ele. Conforme a água (ou a corrente elétrica) entra no resistor, o cano se torna mais fino e o volume da água (corrente) saindo na outra ponta é, dessa forma, reduzido. Você utiliza resistores para diminuir a voltagem ou a corrente para outros dispositivos.

O valor de resistência é conhecido como ohm, e seu símbolo é o ômega grego, Ω. Nesse caso, o pino digital 10 está emitindo 5 V de corrente contínua a 40 mA (miliampères; amperagem de acordo com o datasheet[1] da Atmega), e seu LED requer (de acordo com o datasheet) uma voltagem de 2 V e uma corrente de 35 mA. Portanto, você necessita de um resistor que reduza os 5 V para 2 V, e a corrente de 40 mA para 35 mA, caso queira exibir o LED com brilho máximo. Se você deseja um LED de menor luminosidade, pode utilizar um valor mais alto de resistência.

> **Nota**: NUNCA utilize um valor de resistor mais BAIXO que o necessário. Você colocará corrente demais no LED, danificando-o permanentemente. Você também poderia danificar outros componentes de seu circuito.

A fórmula para calcular o resistor necessário é

$R = (V_S - V_L) / I$

1 N.T.: Datasheet, ou folha de dados, é um termo técnico usado para identificar um documento relativo a um determinado produto, contendo suas especificações técnicas (fonte: Wikipédia).

Em que V_S é a voltagem fornecida, V_L é a voltagem do LED e I é a corrente do LED. Nosso LED de exemplo tem uma voltagem de 2 V e uma corrente de 35 mA, conectado a um pino digital do Arduino, de 5 V, assim o valor necessário para o resistor seria de

R = (5 – 2) / 0.035

o que dá um valor de 85,71.

Resistores vêm com valores-padrão e o valor comum mais próximo nesse caso seria de 100 Ω. Sempre escolha o resistor com valor mais próximo MAIOR do que o valor necessário. Se você escolher um valor menor, muita corrente atravessará o resistor, danificando-o.

Mas como encontrar um resistor de 100 Ω? Um resistor é pequeno demais para conter informações de fácil leitura, por isso, resistores utilizam um código de cores. Ao redor do resistor você tipicamente encontrará quatro faixas de cores; utilizando o código da tabela 2.1 você pode descobrir o valor de um resistor. Da mesma forma, você pode descobrir o código de cores de uma determinada resistência.

Tabela 2.1 – Código de cores dos resistores

Cor	Primeira faixa	Segunda faixa	Terceira faixa (multiplicador)	Quarta faixa (tolerância)
Preto	0	0	$x10^0$	
Marrom	1	1	$x10^1$	±1%
Vermelho	2	2	$x10^2$	±2%
Laranja	3	3	$x10^3$	
Amarelo	4	4	$x10^4$	
Verde	5	5	$x10^5$	±0,5%
Azul	6	6	$x10^6$	±0,25%
Violeta	7	7	$x10^7$	±0,1%
Cinza	8	8	$x10^8$	±0,05%
Branco	9	9	$x10^9$	
Dourado			$x10^{-1}$	±5%
Prata			$x10^{-2}$	±10%
Nenhuma				±20%

De acordo com a tabela, para um resistor de 100 Ω você necessita de 1 na primeira faixa, que é marrom, seguido por um 0 na faixa seguinte, que é preta. Então, deve multiplicar isso por 10^1 (em outras palavras, adicionar um zero), o que resulta em marrom na terceira faixa. A faixa final indica a tolerância do resistor. Caso seu resistor tenha uma faixa dourada, ele tem uma tolerância de ±5%; isso significa que o valor, de fato, do resistor varia entre 95 Ω e 105 Ω. Dessa forma, se você tem um LED que requer 2 V e 35 mA, necessitará de um resistor com uma combinação de faixas Marrom, Preto, Marrom.

Caso queira um resistor de 10 kΩ (ou 10 quilo-ohms), necessitará de uma combinação Marrom, Preto, Laranja (1, 0, +3 zeros). Se necessitar de um resistor de 570 kΩ, as cores serão Verde, Violeta e Amarelo.

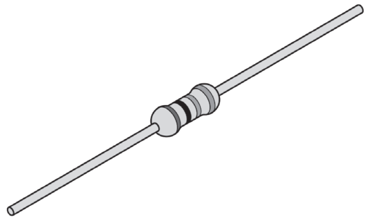


Figura 2.4 – Resistor de 10 kΩ com tolerância de 5%.

Da mesma forma, se você encontrasse um resistor e quisesse saber seu valor, poderia fazer o mesmo processo em ordem inversa. Assim, se encontrasse o resistor da figura 2.4 e quisesse descobrir seu valor para que pudesse guardá-lo em uma caixa devidamente marcada, poderia consultar a tabela e ver que ele tem um valor de 220 Ω.

Agora que você sabe como funciona a codificação por cores, escolha o valor de resistência correto para o LED que você comprou, para completar este projeto.

O componente final (fora os fios jumper, mas estou certo de que você pode descobrir sozinho o que eles fazem) é o LED, que significa Light Emitting Diode (Diodo Emissor de Luz). Um diodo é um dispositivo que permite o fluxo de corrente em apenas uma direção; é como uma válvula em um sistema de distribuição de água, mas nesse caso ele permite o fluxo da corrente elétrica em uma direção. Caso a corrente tente reverter e retornar na direção oposta, o diodo impede que ela o faça. Diodos podem ser úteis para prevenir que alguém conecte acidentalmente a alimentação e o terra aos terminais errados em um circuito, danificando os componentes.

Um LED é a mesma coisa, mas ele também emite luz. LEDs vêm de todos os tipos de cores e níveis de luminosidade, incluindo a parte ultravioleta e infravermelha do espectro (como nos LEDs do controle remoto de sua TV).

Caso examine cuidadosamente seu LED, você perceberá dois detalhes: os terminais têm comprimentos diferentes, e um lado do LED é chanfrado, em vez de cilíndrico (Figura 2.5). Essas são pistas que indicam qual terminal é o ânodo (positivo) e qual é o cátodo (negativo): o terminal mais comprido (ânodo) é conectado à alimentação de energia positiva (3,3 V) e o terminal com o lado chanfrado (cátodo) vai para o terra.

Se você conectar seu LED da forma errada, isso não o danificará (a menos que você coloque correntes muito elevadas nele). Entretanto, é essencial que você sempre coloque um resistor em série com o LED, para garantir que a corrente certa chegue ao LED. Você pode danificar permanentemente o LED se não o fizer.

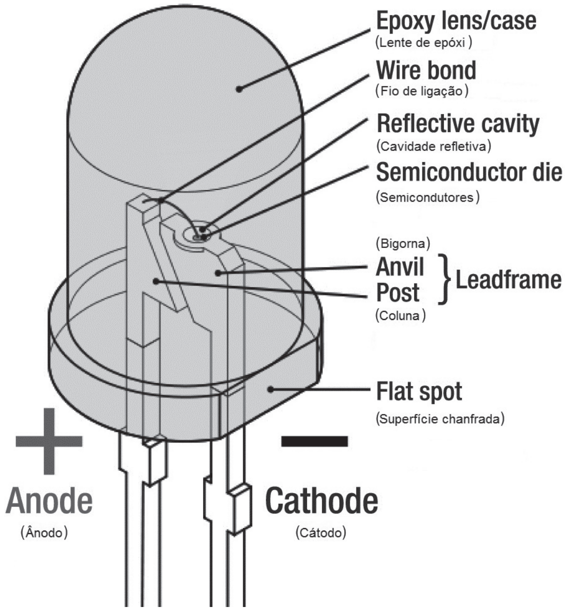


Figura 2.5 – Componentes de um LED (imagem cortesia de Inductiveload do Wikimedia Commons).

Note que você também pode obter LEDs bicolores ou tricolores, os quais têm diversos terminais saindo deles. Um LED RGB oferece um LED vermelho, verde e azul (daí RGB, de Red, Green e Blue) em um único pacote. Esse LED tem quatro terminais; um será um ânodo ou cátodo usual (comum a todos os três LEDs), e os outros terminais irão para o ânodo ou cátodo de um LED individual. Ajustando os valores de brilho dos canais R, G e B do LED RGB você pode obter a cor que quiser (o mesmo efeito pode ser obtido se você utilizar três LEDs separados, sendo um vermelho, um verde e outro azul).

Agora que você sabe como funcionam os componentes e o código deste projeto, vamos experimentar algo um pouco mais interessante.

Projeto 2 – Sinalizador de código Morse S.O.S.

Para este projeto, você reutilizará o circuito que preparamos para o projeto 1 (por isso não será necessário analisar o hardware), mas você utilizará um código diferente para fazer com que o LED sinalize as letras S.O.S., sinal de socorro internacional em código Morse. O código Morse é um tipo de codificação de caracteres que transmite letras e números utilizando padrões de ligado e desligado. Portanto, ele é muito adequado ao seu sistema digital, uma vez que você pode acender e apagar o LED no padrão necessário para soletrar uma palavra ou série de caracteres. Nesse caso, o padrão S.O.S. é formado de três pontos (sinais curtos), seguidos por três traços (sinais longos), seguidos por três pontos novamente.

Para piscar o LED, acendendo e apagando nesse padrão e sinalizando S.O.S., utilize o código da listagem 2.2.

Listagem 2.2 – Código para o projeto 2

```
// LED conectado ao pino 10
int ledPin = 10;

// executa uma vez, quando o sketch inicia
void setup() {
  // define o pino como saída
  pinMode(ledPin, OUTPUT);
}

// executa repetidas vezes
void loop() {

  // 3 pontos
  for (int x=0; x<3; x++) {
    digitalWrite(ledPin, HIGH);  // acende o LED
    delay(150);                  // espera 150ms
    digitalWrite(ledPin, LOW);   // apaga o LED
    delay(100);                  // espera 100ms
  }

  // espera 100ms para marcar o intervalo entre as letras
  delay(100);

  // 3 traços
  for (int x=0; x<3; x++) {
    digitalWrite(ledPin, HIGH);  // acende o LED
    delay(400);                  // espera 400ms
    digitalWrite(ledPin, LOW);   // apaga o LED
    delay(100);                  // espera 100ms
  }

  // espera 100ms para marcar o intervalo entre as letras
  delay(100);

  // 3 pontos novamente
  for (int x=0; x<3; x++) {
    digitalWrite(ledPin, HIGH);  // acende LED
    delay(150);                  // espera 150ms
    digitalWrite(ledPin, LOW);   // apaga o LED
    delay(100);                  // espera 100ms
  }

  // espera 5 segundos antes de repetir o sinal de SOS
  delay(5000);
}
```

Crie um novo sketch e, então, digite o código da listagem 2.2. Verifique se o código está livre de erros e, em seguida, faça seu upload para o Arduino. Se tudo der certo, você verá o LED piscar o código Morse para o sinal de S.O.S., aguardar cinco segundos e, então, repetir o processo.

Se você montasse um Arduino a bateria, com uma luz muito brilhante, e colocasse esse experimento em uma caixa portátil à prova d'água, sua criação poderia ser utilizada para controlar uma luz estroboscópica S.O.S. de emergência, perfeita para ser utilizada em barcos, escaladas etc.

Vamos entender como funciona esse código.

Projeto 2 – Sinalizador de código Morse S.O.S. – Análise do código

A primeira parte do código é idêntica à do último projeto, na qual você inicializa uma variável e define o pino digital 10 como saída. No loop principal do código, você pode ver os mesmos tipos de instruções para acender e apagar o LED de acordo com determinado intervalo de tempo. Dessa vez, entretanto, as instruções estão dentro de três blocos separados de código.

O primeiro bloco é responsável pela saída dos três pontos:

```
for (int x=0; x<3; x++) {
  digitalWrite(ledPin, HIGH);
  delay(150);
  digitalWrite(ledPin, LOW);
  delay(100);
}
```

Você pode ver que o LED fica aceso por 150ms e depois apagado por 100ms; também pode ver que essas instruções estão dentro de um par de chaves e são, portanto, um bloco separado de código. Mas, quando você executa o sketch, vê a luz piscar três vezes, e não apenas uma.

Isso é feito utilizando o loop `for`:

```
for (int x=0; x<3; x++) {
```

É essa instrução que faz com que o código dentro do bloco seja executado três vezes. Há três parâmetros que você deve dar ao loop `for`, que correspondem à inicialização, à condição e ao incremento. A *inicialização* ocorre primeiro e somente uma vez. Cada vez que o loop é percorrido, a *condição* é testada; se for verdadeira, o bloco de instrução e o *incremento* são executados; então, a *condição* é testada novamente. Quando a *condição* se torna falsa, o loop termina.

Assim, primeiro você tem de inicializar uma variável como número inicial do loop. Nesse caso, você utiliza a variável `x`, definindo-a como zero:

```
int x=0;
```

Depois, você define uma condição para decidir quantas vezes o código no loop será executado:

```
x<3;
```

Nesse caso, o código fará o loop se `x` for menor que (<) 3.

O símbolo < é o que chamamos de *operador de comparação*, utilizado para tomar decisões dentro do código e comparar dois valores. Os símbolos utilizados são:

- `==` (igual a)
- `!=` (não igual a)
- `<` (menor que)
- `>` (maior que)
- `<=` (menor ou igual a)
- `>=` (maior ou igual a)

Em seu código, você está comparando `x` ao valor `3`, para ver se ele é menor do que 3. Se isso for verdade, o código no bloco repetirá.

A instrução final

```
x++
```

É uma instrução que eleva o valor de `x` em 1 unidade. Você também poderia ter digitado `x = x + 1`, o que atribuiria a `x` o valor de `x + 1`. Note que não é necessário colocar um ponto e vírgula depois dessa instrução final no loop `for`.

Você pode realizar operações matemáticas simples utilizando os símbolos `+`, `-`, `*` e `/` (adição, subtração, multiplicação e divisão). Por exemplo:

- 1 + 1 = 2
- 3 - 2 = 1
- 2 * 4 = 8
- 8 / 2 = 4

Assim, seu loop `for` inicializa o valor de `x` como `0`, e depois executa o código dentro do bloco (dentro das chaves). Então, ele adiciona o incremento (nesse caso, adiciona `1` a `x`). Por fim, ele verifica se a condição é válida (se `x` é menor que `3`) e, se esse for o caso, repete o processo.

Agora que você sabe como funciona o loop `for`, pode ver que há três loops `for` em seu código: um que faz um loop três vezes e exibe os pontos, outro que repete três vezes e exibe os traços e, por fim, uma repetição dos pontos.

Repare que a variável `x` tem *escopo local*, o que significa que pode ser vista apenas pelo código dentro de seu próprio bloco. Do contrário, se ela for inicializada fora da função `setup()`,terá um *escopo global* e poderá ser vista pelo programa inteiro. Em nosso caso, se você tentar acessar `x` fora do loop `for`, receberá um erro.

Entre cada loop `for` há uma pequena espera, para criar uma leve pausa entre as letras do S.O.S. Por fim, o código aguarda cinco segundos, antes que o loop principal do programa reinicie desde o início.

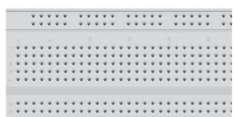
Agora, vamos passar para o uso de LEDs múltiplos.

Projeto 3 – Semáforo

Neste projeto, você criará um semáforo que irá do verde ao vermelho, passando pelo amarelo, e que retornará depois de um intervalo de tempo, utilizando o sistema de quatro estados do Reino Unido. Este projeto poderia ser utilizado para criar um conjunto funcional de semáforos para uma maquete de ferrovia ou para uma pequena cidade de brinquedo. Caso você não seja do Reino Unido, pode modificar o código e as cores de acordo com os sinais de seu país. Primeiro, entretanto, crie o projeto como instruído, e faça as alterações apenas depois de saber como tudo funciona.

Componentes necessários

Protoboard



LED vermelho difuso



LED amarelo difuso



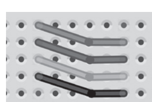
LED verde difuso



3 resistores de 150 ohms*



Fios jumper



* Ou o valor apropriado para seu LED.

Conectando os componentes

Conecte seu circuito como mostra a figura 2.6. Dessa vez, você conectará três LEDs, com o ânodo de cada um indo para os pinos digitais 8, 9 e 10, por meio de um resistor de 150 Ω cada (ou do valor necessário para seu caso).

Leve um fio jumper do terra do Arduino para o barramento do terra no topo da protoboard; um fio terra vai do terminal cátodo de cada LED para o barramento terra comum por meio de um resistor — dessa vez conectado ao cátodo. (Para esse circuito simples, não importa se o resistor está conectado ao ânodo ou ao cátodo).

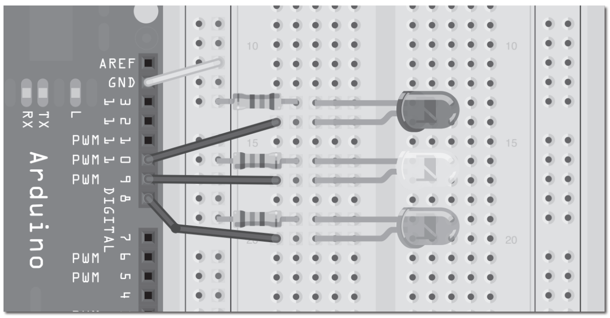


Figura 2.6 – Circuito para o Projeto 3 – Semáforo (consulte o site da Novatec para versão colorida).

Digite o código

Digite o código da listagem 2.3, verifique-o, e faça o upload para seu Arduino. Os LEDs agora atravessarão quatro estados que simulam o sistema de semáforos do Reino Unido (Figura 2.7). Caso você tenha acompanhado os projetos 1 e 2, tanto o código quanto o hardware do projeto 3 devem ser evidentes. Deixarei que você analise o código e descubra como ele funciona.

Listagem 2.3 – Código para o projeto 3

```
// Projeto 3 - Semáforo

int ledDelay = 10000;    // espera entre as alterações
int redPin = 10;
int yellowPin = 9;
int greenPin = 8;

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}
```

```
void loop() {

  digitalWrite(redPin, HIGH);      // acende a luz vermelha
  delay(ledDelay);                 // espera 5 segundos

  digitalWrite(yellowPin, HIGH);   // acende a luz amarela
  delay(2000);                     // espera 2 segundos

  digitalWrite(greenPin, HIGH);    // acende a luz verde
  digitalWrite(redPin, LOW);       // apaga a luz vermelha
  digitalWrite(yellowPin, LOW);    // apaga a luz amarela
  delay(ledDelay);                 // espera ledDelay milissegundos

  digitalWrite(yellowPin, HIGH);   // acende a luz amarela
  digitalWrite(greenPin, LOW);     // apaga a luz verde
  delay(2000);                     // espera 2 segundos

  digitalWrite(yellowPin, LOW);    // apaga a luz amarela
                                   // agora nosso loop se repete
}
```

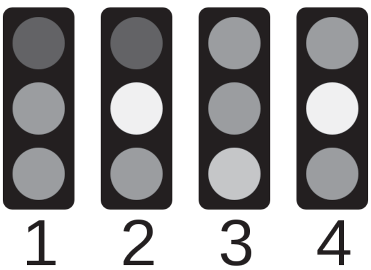


Figura 2.7 – Quatro estados do sistema de semáforos do Reino Unido (imagem por Alex43223 do WikiMedia) (consulte o site da Novatec para versão colorida).

Projeto 4 – Semáforo interativo

Dessa vez, você estenderá o projeto anterior para incluir um farol de pedestre e um botão, que será pressionado pelos pedestres para solicitar a travessia da rua. O Arduino reagirá quando o botão for pressionado, alterando o estado das luzes para que os carros parem e os pedestres possam atravessar em segurança.

Essa será a primeira vez que você vai interagir com o Arduino, fazendo-o realizar algo quando você alterar o estado de um botão que está sendo observado. Neste projeto, você também aprenderá como criar suas próprias funções dentro do código.

Daqui em diante, não listarei mais a protoboard, nem os fios jumper, na lista de componentes necessários. Note que você sempre terá de utilizá-los.

Componentes necessários

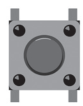
2 LEDs vermelhos difusos 

LED amarelo difuso 

2 LEDs verdes difusos 

Resistor de 150 ohms 

4 resistores 

Botão 

Escolha o valor de resistor apropriado para os LEDs utilizados em seu projeto. O resistor de 150 Ω é para o botão; ele é conhecido como um *resistor pull-down* (que será definido posteriormente). O botão é às vezes chamado pelos fornecedores de *interruptor tátil*, e é perfeito para ser utilizado com a protoboard.

Conectando os componentes

Conecte seu circuito como mostra a figura 2.8. Verifique cuidadosamente a fiação antes de ligar seu Arduino. Lembre-se de que o Arduino deve estar desconectado da força enquanto você conecta o circuito.

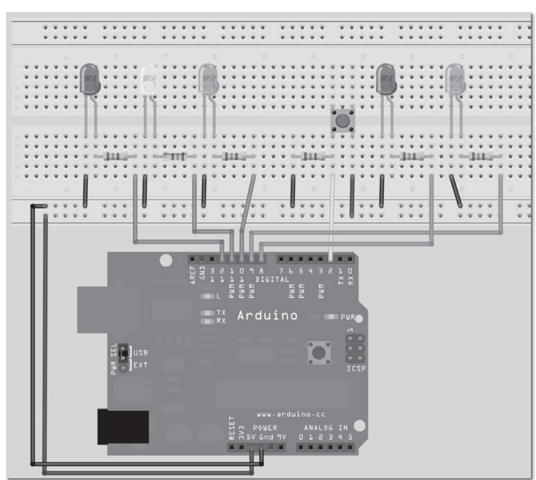


Figura 2.8 – Circuito para o Projeto 4 – Sistema de semáforo com travessia de pedestres e botão de requisição (consulte o site da Novatec para versão colorida).

Digite o código

Digite o código da listagem 2.4, verifique-o e faça seu upload. Quando você executar o programa, ele iniciará com o semáforo no verde, para permitir que os carros passem, e a luz para pedestres no vermelho.

Quando você pressionar o botão, o programa verificará se, ao menos, cinco segundos transcorreram desde a última vez em que o semáforo mudou (para permitir que o trânsito flua) e, se afirmativo, executará o código para a função que você criou, `changeLights()`. Nessa função, o semáforo vai de verde para amarelo e depois para vermelho, e o semáforo para pedestres vai para verde. Depois de um intervalo de tempo definido na variável `crossTime` (tempo suficiente para que os pedestres atravessem), a luz verde para pedestres pisca, acendendo e apagando, para avisar aos pedestres que atravessem rapidamente antes que o semáforo feche. Então, a luz vermelha do semáforo para pedestres acende, e a luz para os veículos vai de vermelho para amarelo, e finalmente para verde, permitindo novamente o fluxo do tráfego.

Listagem 2.4 – Código para o projeto 4

```
// Projeto 4 – Semáforo interativo

int carRed = 12;          // estabelece o semáforo para carros
int carYellow = 11;
int carGreen = 10;
int pedRed = 9;           // estabelece o semáforo para pedestres
int pedGreen = 8;
int button = 2;           // pino do botão
int crossTime = 5000;       // tempo para que os pedestres atravessem
unsigned long changeTime;   // tempo desde que o botão foi pressionado

void setup() {
  pinMode(carRed, OUTPUT);
  pinMode(carYellow, OUTPUT);
  pinMode(carGreen, OUTPUT);
  pinMode(pedRed, OUTPUT);
  pinMode(pedGreen, OUTPUT);
  pinMode(button, INPUT);    // botão no pino 2
  // acende a luz verde
  digitalWrite(carGreen, HIGH);
  digitalWrite(pedRed, HIGH);
}

void loop() {
  int state = digitalRead(button);
  /* verifica se o botão foi pressionado e se transcorreram 5 segundos desde a última vez que
     isso ocorreu */
  if (state == HIGH && (millis() - changeTime) > 5000) {
```

```
      // Chama a função para alterar as luzes
      changeLights();
    }
  }
  void changeLights() {
    digitalWrite(carGreen, LOW);      // apaga o verde
    digitalWrite(carYellow, HIGH);    // acende o amarelo
    delay(2000);    // espera 2 segundos
    digitalWrite(carYellow, LOW);      // apaga o amarelo
    digitalWrite(carRed, HIGH);       // acende o vermelho
    delay(1000);  // espera 1 segundo, por segurança

    digitalWrite(pedRed, LOW);        // apaga o vermelho dos pedestres
    digitalWrite(pedGreen, HIGH);     // acende o verde dos pedestres
    delay(crossTime);    // espera por um intervalo de tempo predefinido
    // pisca o verde dos pedestres
    for (int x=0; x<10; x++) {
      digitalWrite(pedGreen, HIGH);
      delay(250);
      digitalWrite(pedGreen, LOW);
      delay(250);
    }
    // acende o vermelho dos pedestres
    digitalWrite(pedRed, HIGH);
    delay(500);
    digitalWrite(carYellow, HIGH);     // acende o amarelo
    digitalWrite(carRed, LOW);        // apaga o vermelho
    delay(1000);
    digitalWrite(carGreen, HIGH);     // acende o verde
    digitalWrite(carYellow, LOW);     // apaga o amarelo
    // registra o tempo desde a última alteração no semáforo
    changeTime = millis();
    // depois retorna para o loop principal do programa
  }
```

Projeto 4 – Semáforo interativo – Análise do código

Com base nos projetos anteriores, você já deve compreender e reconhecer a maioria do código deste projeto. Destacarei apenas as novas palavras-chave e conceitos.

```
unsigned long changeTime;
```

Aqui temos um novo tipo de dados para uma variável. Anteriormente, você criou tipos de dados inteiros, capazes de armazenar um número entre -32.768 e 32.767. Dessa vez,

você criou um tipo de dado `long`, que pode armazenar um número entre -2.137.483.648 e 2.147.483.647. Entretanto, você especificou um `unsigned long`, o que significa que a variável não pode armazenar números negativos, assim o intervalo vai de 0 a 4.294.967.295. Se você estivesse utilizando um inteiro para armazenar o intervalo de tempo desde a última troca no semáforo, teria um tempo máximo de apenas 32 segundos, antes que a variável atingisse um número maior do que o valor que ela pode armazenar.

Como é improvável que um pedestre sempre atravesse a cada 32 segundos, você não quer que o programa deixe de funcionar quando sua variável "transbordar" ao tentar armazenar um número grande demais para seu tipo de dados. Assim, você utiliza um tipo de dados `unsigned long` para obter um grande intervalo de tempo entre pressionamentos de botão

- 4.294.967.295 milissegundos = 4.294.967 segundos
- 4.294.967 segundos = 71.582 minutos
- 71.582 minutos = 1.193 horas
- 1.193 horas = 49 dias

É praticamente inevitável que o botão para o semáforo de pedestres seja pressionado ao menos uma vez a cada 49 dias, por isso você não deve ter problemas com esse tipo de dado.

Então, por que não há apenas um único tipo de dados, capaz de sempre armazenar números enormes? Bem, porque variáveis ocupam espaço na memória; quanto maior o número, mais memória será utilizada para armazenar as variáveis. Em seu PC ou laptop, você não terá de se preocupar muito com isso, mas em um pequeno microcontrolador, como o Atmega32 do Arduino, é essencial que você utilize apenas o menor tipo de dado de variável disponível para seu propósito.

A tabela 2.2 lista os vários tipos de dados que você pode utilizar em seus sketches.

Tabela 2.2 – Tipos de dados

Tipo de dados	RAM	Intervalo numérico
void keyword	N/A	N/A
boolean	1 byte	0 a 1 (false ou true)
byte	1 byte	0 a 255
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255
int	2 bytes	-32.768 a 32.767
unsigned int	2 bytes	0 a 65.535
word	2 bytes	0 a 65.535
long	4 bytes	-2.147.483.648 a 2.147.483.647

Tipo de dados	RAM	Intervalo numérico
unsigned long	4 bytes	0 a 4.294.967.295
float	4 bytes	-3,4028235E+38 a 3,4028235E+38
double	4 bytes	-3,4028235E+38 a 3,4028235E+38
string	1 byte + x	Sequência de caracteres
array	1 byte + x	Coleção de variáveis

Cada tipo de dado utiliza determinada quantidade de memória: algumas variáveis utilizam apenas 1 byte de memória, enquanto outras, 4 bytes ou mais (não se preocupe em saber o que é um byte por enquanto; discutiremos isso futuramente). Note que você não pode copiar dados de um tipo de dados para outro. Em outras palavras, se `x` for um `int` e `y` uma `string`, `x = y` não dará certo, pois os tipos de dados são diferentes.

O Atmega168 tem 1 kB (1.000 bytes) e o Atmega328, 2 kB (2.000 bytes) de SRAM; isso não é muita memória. Em programas de tamanho considerável, com muitas variáveis, você poderá facilmente ficar sem memória se não empregar os tipos de dados corretos. Como você utilizou `int` (que usa até 2 bytes, e pode armazenar um número de até 32.767) para armazenar o número de seu pino, que poderia ser no máximo 13 em seu Arduino (e 54 no Arduino Mega), acabou consumindo mais memória do que o necessário. Poderíamos ter economizado memória utilizando o tipo de dados *byte*, que pode armazenar um número entre 0 e 255 — intervalo mais do que suficiente para armazenar o número de um pino de entrada/saída.

Em seguida, temos

```
pinMode(button, INPUT);
```

Isso diz ao Arduino que você deseja utilizar o pino digital 2 (`button = 2`) como `INPUT`. Você utilizará o pino digital 2 para escutar quando o botão é pressionado, por isso seu modo deve ser definido como entrada.

No loop principal do programa, você verifica o estado do pino 2 com esta instrução:

```
int state = digitalRead(button);
```

Isso inicializa um inteiro (sim, é um desperdício, e seria melhor utilizar um booleano), `state`, e então define seu valor como o valor do pino digital 2. A instrução `digitalRead` lê o estado do pino dentro dos parênteses, retornando esse estado para o valor inteiro ao qual você o atribuiu. Depois, você pode verificar o valor em `state` para ver se o botão foi ou não pressionado:

```
if (state == HIGH && (millis() - changeTime) > 5000) {
  // Chama a função para alterar as luzes
  changeLights();
  }
```

A instrução `if` é um exemplo de uma estrutura de controle, cujo propósito é verificar se determinada condição foi ou não atingida. Caso a condição seja verdadeira, a instrução executa o código dentro de seu bloco de código. Por exemplo, se você quisesse acender um LED, caso uma variável `x` atingisse um valor superior a `500`, poderia escrever o seguinte:

```
if (x>500) {digitalWrite(ledPin, HIGH);
```

Quando você lê um pino utilizando o comando `digitalRead`, o estado do pino será `HIGH` ou `LOW`. Assim, o comando `if` em seu sketch tem a seguinte apresentação:

```
if (state == HIGH && (millis() - changeTime) > 5000)
```

Aqui, verificamos se duas condições foram atingidas. A primeira é a de que a variável `state` seja `HIGH`. Se o botão foi pressionado, `state` será `HIGH`, pois você já definiu este como o valor a ser lido do pino digital 2. Você também está conferindo se o valor de `millis() - changeTime` é maior que `5000` (utilizando o comando lógico E, `&&`). `millis()` é uma função integrada à linguagem do Arduino, que retorna o número de milissegundos desde que o Arduino iniciou a execução do programa atual. Sua variável `changeTime` inicialmente não armazenará nenhum valor, mas depois que a função `changeLights()` tiver sido executada, `changeTime` será definida como o valor atual de `millis()`.

Subtraindo o valor na variável `changeTime` do valor atual de `millis()`, você pode verificar se transcorreram cinco segundos desde que `changeTime` foi definida pela última vez. O cálculo de `millis() - changeTime` é colocado dentro de seus próprios parênteses, para garantir que você compare o valor de `state` ao resultado dessa operação, e não ao valor de `millis()` por si só.

O símbolo `&&` que vemos entre

```
state == HIGH
```

e o cálculo é um exemplo de um operador booleano. Nesse caso, ele significa E. Para entender o que isso quer dizer, vamos analisar todos os operadores booleanos:

- `&&` E lógico
- `||` OU lógico;
- `!` NÃO

Note que essas são instruções lógicas, podendo ser utilizadas para testar diversas condições em instruções `if`.

`&&` é verdadeiro apenas se ambos os operandos forem verdadeiros, assim, a instrução `if` a seguir executará seu código apenas se `x` for `5`, e `y` for `10`:

```
if (x==5 && y==10)  {....
```

`||` é verdadeiro se um ou mais operandos forem verdadeiros; por exemplo, a instrução `if` a seguir será executada se `x` for `5` ou se `y` for `10`:

```
if (x==5 || y==10) {.....
```

A instrução `!` (NÃO) é verdadeira apenas se o operando for falso; assim, a instrução `if` a seguir será executada se `x` for falso, ou seja, igual a zero:

```
if (!x) {.......
```

Você também pode *aninhar* condições entre parênteses, por exemplo:

```
if (x==5 && (y==10 || z==25)) {.......
```

Nesse caso, as condições dentro dos parênteses são processadas separadamente, e tratadas como uma única condição. Depois, são comparadas com a condição fora dos parênteses. Assim, se você esboçar uma simples *tabela-verdade* (Tabela 2.3) para essa instrução, poderá ver como isso funciona.

Tabela 2.3 – Tabela de verdade para a condição (`x==5 && (y == 10 || z == 25)`*)*

x	y	z	Verdadeiro/Falso
4	9	25	FALSO
5	10	24	VERDADEIRO
7	10	25	FALSO
5	10	25	VERDADEIRO

O comando dentro da instrução `if` é

```
changeLights();
```

Esse é um exemplo de uma chamada de função. Uma *função* é simplesmente um bloco de código separado que recebeu um nome. Entretanto, funções também podem receber parâmetros e/ou retornar dados. Nesse caso, você não passou nenhum dado à função, nem fez com que ela retornasse dados. No futuro, falarei mais detalhadamente sobre a transmissão de parâmetros e o retorno de dados em funções.

Quando `changeLights()` é chamada, a execução do código pula da linha atual para a função, executa o código dentro dela, e depois retorna ao ponto no código, logo depois da chamada à função.

Nesse caso, se as condições da instrução `if` forem atendidas, o programa executará o código dentro da função e retornará para a linha seguinte, depois de `changeLights()`, na instrução `if`.

O código dentro da função simplesmente altera o semáforo dos veículos para vermelho, passando pelo amarelo, e então acende a luz verde para pedestres. Depois de um intervalo de tempo definido pela variável `crossTime`, a luz dos pedestres pisca algumas vezes para alertar que o tempo está se esgotando, e então passa para o vermelho. Enquanto isso, o semáforo dos veículos muda de vermelho para verde, passando pelo amarelo e retornando, dessa forma, ao seu estado normal.

O loop principal do programa verifica continuamente se o botão para pedestres foi ou não pressionado. Se isso ocorrer *e* (`&&`) o tempo desde a última alteração do semáforo for maior que 5 segundos, o loop chamará novamente a função `changeLights()`.

Nesse programa, não houve nenhum benefício em colocar o código dentro de suas próprias funções, além de conseguir uma melhor apresentação e facilitar a explicação do conceito das funções. É apenas quando uma função recebe parâmetros e/ou retorna dados que seus verdadeiros benefícios vêm à tona; você verá isso quando utilizar funções novamente.

Projeto 4 – Semáforo interativo – Análise do hardware

O novo componente de hardware apresentado no projeto 4 é o botão, ou interruptor tátil. Como você pode ver, ao analisar o circuito, o botão não está diretamente conectado entre a linha de alimentação e o pino de entrada; há um resistor entre o botão e o barramento do terra. Isso é o que chamamos de um resistor pull-down, elemento essencial para garantir que botões funcionem corretamente. Farei uma breve pausa para explicar resistores pull-up e pull-down.

Estados lógicos

Um *circuito lógico* é um circuito projetado para emitir uma saída de ligado ou desligado, representada pelos números binários 1 e 0, respectivamente. O estado desligado (ou zero) é representado por uma voltagem próxima a zero volt na saída; um estado de ligado (ou 1) é representado por um nível mais alto, próximo à tensão de alimentação. A representação mais simples de um circuito lógico é um interruptor (Figura 2.9).



Figura 2.9 – Símbolo eletrônico para um interruptor.

Quando um interruptor está aberto, nenhuma corrente pode atravessá-lo e nenhuma voltagem pode ser medida na saída. Quando você fecha o interruptor, a corrente pode fluir e uma voltagem pode ser medida na saída. O estado aberto pode ser imaginado como 0 e o estado fechado como 1, em um circuito lógico.

Em um circuito lógico, se a voltagem esperada para representar o estado ligado (ou 1) é de 5 V, é importante que, quando a saída do circuito for 1, a voltagem esteja o mais próximo possível desse valor. Se você não garantir que os estados estejam próximos das voltagens necessárias, essa parte do circuito pode ser considerada como *flutuante*

(não representativa de um estado alto ou baixo). Esse estado também é conhecido como ruído elétrico, e ruído em um circuito digital pode ser interpretado como valores aleatórios de 1 e 0.

É aqui que resistores pull-up e pull-down podem ser utilizados para garantir que o estado seja alto ou baixo. Se você deixar que esse nó no circuito flutue, ele poderá ser interpretado aleatoriamente como 0 ou como 1, o que não é desejável. É preferível forçá-lo para o estado desejado.

Resistores pull-down

A figura 2.10 mostra um esquema em que um resistor pull-down é utilizado. Se o botão for pressionado, a eletricidade tomará o caminho de menor resistência, e fluirá entre os 5 V e o pino de entrada (há um resistor de 100 Ω no pino de entrada e um resistor de 10 kΩ no terra). Entretanto, quando o botão não é pressionado, a entrada está conectada ao resistor de 100 kΩ e é direcionada para o terra. Sem isso, o pino de entrada ficaria conectado a nada quando o botão não estivesse pressionado, e flutuaria entre 0 V e 5 V. Nesse circuito, a entrada sempre será direcionada para o terra, ou zero volt, quando o botão não estiver pressionado, e para 5 V quanto ele estiver pressionado. Em outras palavras, com isso você impede que o pino flutue entre dois valores.

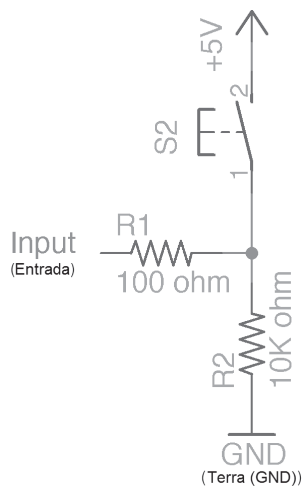


Figura 2.10 – Circuito de resistor pull-down.

Agora, veja a figura 2.11.

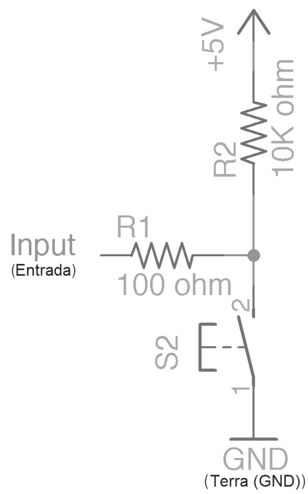


Figura 2.11 – Circuito de resistor pull-up.

Resistores pull-up

Nesse circuito você trocou as posições do resistor pull-down e do interruptor. O resistor agora se torna um resistor pull-up. Como você pode ver, quando o botão não estiver pressionado, o pino de entrada será ligado aos 5 V, e estará sempre alto. Quando o botão for pressionado, o caminho de menor resistência será em direção ao terra, por isso o pino será ligado ao terra, ou o estado baixo. Sem o resistor entre os 5 V e o terra, isso seria um curto-circuito, o que danificaria seu circuito ou sua fonte de alimentação. Graças ao resistor, não temos mais um curto-circuito, pois o resistor limita a quantidade de corrente. O resistor pull-up é mais comumente utilizado em circuitos digitais.

Utilizando resistores pull-up e pull-down simples, você pode garantir que o estado de um pino de entrada seja sempre alto ou baixo, dependendo de sua aplicação.

No projeto 4, você utiliza um resistor pull-down para garantir que o botão seja corretamente registrado pelo Arduino. Vamos analisar novamente o resistor pull-down desse circuito (Figura 2.12).

Esse circuito contém um botão. Um pino do botão está conectado diretamente aos 5 V, e outro diretamente ao pino digital 2. Ele também está conectado diretamente ao terra, por meio de um resistor pull-down. Isso significa que quando o botão não está pressionado, o pino é ligado ao terra e, portanto, lê um estado zero, ou baixo. Quando o botão é pressionado, 5 volts fluem para o pino, e ele é lido como um estado 1, ou alto. Detectando se a entrada é alta ou baixa, você pode detectar se o botão está ou não pressionado. Se o resistor não estivesse presente, o fio do pino de entrada

não estaria conectado a nada e estaria flutuando. O Arduino poderia ler isso como um estado `HIGH` ou `LOW`, o que poderia resultar em falsos registros de botões apertados.

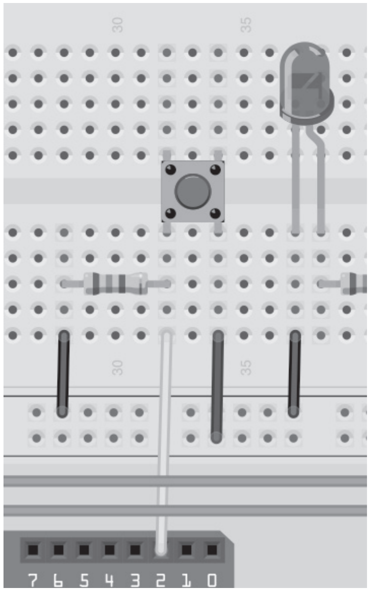


Figura 2.12 – Resistor pull-down do Projeto 4 (consulte o site da Novatec para versão colorida).

Resistores pull-up são muitas vezes utilizados em circuitos digitais para garantir que uma entrada seja mantida alta. Por exemplo, o Circuito Integrado do Registrador de Deslocamento 74HC595, que você utilizará futuramente neste livro, tem um pino de Master Reset. Esse pino reinicia o chip quando ele está em estado baixo. Como resultado, é essencial que o pino seja mantido alto a todo momento, a menos que você queira especificamente reinicializar; você pode manter esse pino no estado alto utilizando sempre um resistor pull-up. Quando quiser reinicializá-lo, pode alterá-lo para o estado baixo utilizando uma saída digital definida como LOW; em todos os outros casos, ele permanecerá alto. Muitos outros Circuitos Integrados têm pinos que devem ser mantidos no estado alto na maioria dos casos, e apenas alterados para o estado baixo para ativar diversas funções.

Resistores pull-up internos do Arduino

Convenientemente, o Arduino contém resistores pull-up que estão conectados aos pinos (os pinos analógicos também têm resistores pull-up). Esses resistores têm um valor de 20 kΩ e têm de ser ativados via software para que possam ser utilizados. Para ativar um resistor pull-up em um pino, você tem de, primeiro, alterar seu `pinMode` para `INPUT` e escrever um `HIGH` nesse pino, utilizando um comando `digitalWrite`:

```
pinMode(pin, INPUT);
digitalWrite(pin, HIGH);
```

Caso você altere o `pinMode` de `INPUT` para `OUTPUT` depois de ativar os resistores pull-up internos, o pino permanecerá em um estado `HIGH`. Isso também funciona de forma inversa: um pino de saída que está em um estado `HIGH`, e que é posteriormente trocado para um modo `INPUT`, terá seus resistores pull-up internos habilitados.

Resumo

Seus quatro primeiros projetos abordaram muitos tópicos. Você agora sabe o básico sobre como ler entradas e acender e apagar LEDs; da mesma forma, você está começando a construir seu conhecimento sobre eletrônica, compreendendo como funcionam LEDs e resistores, como resistores podem ser utilizados para limitar correntes e como podem alterar uma entrada para um estado alto ou baixo, de acordo com suas necessidades. Você também já deve ser capaz de escolher um resistor, e descobrir seu valor em ohms, analisando suas faixas coloridas. Sua compreensão da linguagem de programação do Arduino está bem encaminhada, e você foi apresentado a muitos comandos e conceitos.

As habilidades aprendidas no capítulo 2 são fundamentais até mesmo para os projetos mais complexos do Arduino. No capítulo 3, continuaremos utilizando LEDs para criar diversos efeitos e, nesse processo, aprenderemos muitos novos comandos e conceitos. Esse conhecimento preparará você para assuntos mais avançados que abordaremos futuramente no livro.

Assuntos e conceitos abordados no capítulo 2:

- a importância dos comentários no código;
- variáveis e seus tipos;
- o propósito das funções `setup()` e `loop()`;
- o conceito de funções e como criá-las;
- a definição do `pinMode` de um pino digital;
- como escrever um valor `HIGH` ou `LOW` para um pino;
- como implementar uma espera, por um número específico de milissegundos;
- protoboards e como utilizá-las;
- o que é um resistor, seu valor de medida e como utilizá-lo para limitar a corrente;
- como calcular o valor necessário do resistor para um LED;
- como calcular o valor de um resistor a partir de suas faixas coloridas;
- o que é um LED e como ele funciona;

- como fazer com que o código repita, utilizando um loop `for`;
- operadores de comparação;
- como realizar operações matemáticas simples no código;
- a diferença entre escopo global e local;
- resistores pull-up e pull-down, e como utilizá-los;
- como ler o pressionamento de um botão;
- como tomar decisões, utilizando a instrução `if`;
- como alterar os modos de um pino entre `INPUT` e `OUTPUT`;
- a função `millis()` e como utilizá-la;
- operadores booleanos e como utilizá-los para tomar decisões lógicas.

CAPÍTULO 3

Efeitos com LEDs

No capítulo 2 você aprendeu o básico sobre entradas e saídas, alguns fundamentos de eletrônica e vários conceitos de programação. Neste capítulo, você continuará utilizando LEDs, mas fará com que produzam efeitos mais elaborados. Este capítulo não se concentra muito em eletrônica; em vez disso, você será apresentado a vários conceitos importantes de codificação, como arrays, funções matemáticas e comunicações seriais, que fornecerão as habilidades de programação necessárias para que você enfrente os projetos mais avançados, ainda por vir neste livro.

Projeto 5 – Efeito de iluminação sequencial com LEDs

Você utilizará uma sequência de LEDs (dez no total) para realizar um efeito de iluminação sequencial, semelhante ao do carro KITT, da série de TV "A Super Máquina", ou do rosto dos Cylons da série "Battlestar Galactica". Este projeto apresentará o conceito de arrays.

Componentes necessários

10 LEDs de 5 mm



10 resistores limitadores de corrente



Conectando os componentes

Primeiramente, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, utilize sua protoboard, seus LEDs, resistores e fios para conectar todos os componentes como mostra a figura 3.1. Verifique calmamente seu circuito antes de conectar a força novamente ao Arduino.

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 3.1.

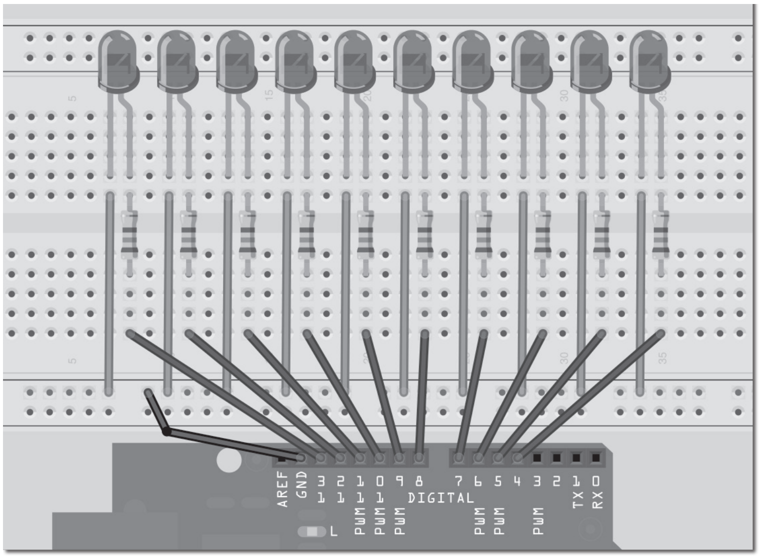


Figura 3.1 – Circuito para o Projeto 5 – Efeito de iluminação sequencial com LEDs (consulte o site da Novatec para versão colorida).

Listagem 3.1 – Código para o Projeto 5

```
// Projeto 5 - Efeito de iluminação sequencial com LEDs
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};   // cria um array para os pinos dos LEDs
int ledDelay(65);  // intervalo entre as alterações
int direction = 1;
int currentLED = 0;
unsigned long changeTime;

void setup() {
  for (int x=0; x<10; x++) {   // define todos os pinos como saída
    pinMode(ledPin[x], OUTPUT); }
  changeTime = millis();
}

void loop() {
  if ((millis() - changeTime) > ledDelay) {    // verifica se já transcorreram ledDelay ms desde
                                               // a última alteração
    changeLED();
    changeTime = millis();
  }
}
void changeLED() {
  for (int x=0; x<10; x++) {   // apaga todos os LEDs
       digitalWrite(ledPin[x], LOW);
  }
```

```
    digitalWrite(ledPin[currentLED], HIGH);  // acende o LED atual
    currentLED += direction; // incrementa de acordo com o valor de direction
    // altera a direção se tivermos atingido o fim
    if (currentLED == 9) {direction = -1;}
    if (currentLED == 0) {direction = 1;}
}
```

Pressione o botão Verify/Compile no topo do IDE, para garantir que não haja erros em seu código. Se não houver problemas, clique no botão Upload. Caso tudo tenha dado certo, os LEDs parecerão se mover, acendendo e apagando sequencialmente e, depois, retornando ao início.

Não apresentei nenhum hardware novo neste projeto, por isso não é necessário analisar novos componentes. Entretanto, apresentei um novo conceito no código, na forma de arrays. Vamos analisar o código do projeto 5 e ver como eles funcionam.

Projeto 5 – Efeito de iluminação sequencial com LEDs – Análise do código

A primeira linha nesse sketch:

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

é uma declaração de variável do tipo array. Um array é um conjunto de variáveis, acessadas utilizando um número de índice. Em seu sketch, você declara um array com o tipo de dado byte, dando ao array o nome `ledPin`. Então, você inicializa-o com dez valores (os pinos digitais 4 a 13). Para acessar um elemento do array, você simplesmente se refere ao número de índice desse elemento. Arrays são indexados a partir de zero, o que significa que o primeiro índice é 0, e não 1. Assim, em seu array de dez elementos, os números de índice vão de 0 a 9. Nesse caso, o elemento 3, (`ledPin[2]`), tem o valor `6`, e o elemento 7, (`ledPin[6]`), tem o valor `10`.

Você deve declarar o tamanho do array, se não for inicializá-lo com dados. Em seu sketch, você não escolheu explicitamente um tamanho, pois o compilador é capaz de contar os valores que você atribuiu ao array e descobrir que seu tamanho é de dez elementos. Caso você tivesse declarado o array, mas não inicializado os valores ao mesmo tempo, teria de declarar um tamanho. Por exemplo, você poderia ter feito o seguinte:

```
byte ledPin[10];
```

e, então, carregado dados nos elementos posteriormente. Para recuperar um valor do array, você faria algo como:

```
x = ledpin[5];
```

Nesse exemplo, `x` agora armazenaria um valor de `8`.

Retornando ao programa, você começou declarando e inicializando um array que armazena dez valores: os pinos digitais utilizados para as saídas de seus dez LEDs.

Em seu loop principal, você verifica se transcorreram ao menos `ledDelay` milissegundos desde a última alteração nos LEDs; em caso afirmativo, o código passa o controle para sua função. O motivo de você passar o controle para a função `changeLED()` dessa maneira, em vez de utilizar comandos `delay()`, é para permitir que, se necessário, outro código seja executado no loop principal do programa (desde que esse código demore menos que `ledDelay` para ser executado).

A função que você cria é

```
void changeLED() {
  // apaga todos os LEDs
  for (int x=0; x<10; x++) {
      digitalWrite(ledPin[x], LOW);
  }
  // acende o LED atual
  digitalWrite(ledPin[currentLED], HIGH);
  // incrementa de acordo com o valor de direction
  currentLED += direction;
  // altera a direção se tivermos atingido o fim
  if (currentLED == 9) {direction = -1;}
  if (currentLED == 0) {direction = 1;}
}
```

O trabalho dessa função é apagar todos os LEDs e, então, acender o LED atual (isso é feito tão rápido que você não verá acontecer), que está armazenado na variável `currentLED`.

Depois, adicionamos `direction` a essa variável. Como `direction` pode ser apenas `1` ou `-1`, o número incrementará (`currentLED +1`) ou decrementará em um (`currentLED+(-1)`).

Em seguida, temos uma instrução `if` para verificar se atingimos o fim da linha de LEDs; se afirmativo, você reverte a variável `direction`.

Alterando o valor de `ledDelay`, você pode fazer com que os LEDs acendam e apaguem sequencialmente em velocidades diferentes. Experimente outros valores para ver os resultados.

Note que você tem de interromper o programa, alterar manualmente o valor de `ledDelay` e, então, fazer o upload do novo código para ver as alterações. Não seria interessante se pudéssemos ajustar a velocidade enquanto o programa está sendo executado? Certamente que sim, e faremos exatamente isso no próximo projeto. Você aprenderá como interagir com o programa e ajustar a velocidade utilizando um potenciômetro.

Projeto 6 – Efeito interativo de iluminação sequencial com LEDs

Deixe intacta sua placa de circuito utilizada no projeto 5. Você apenas adicionará um potenciômetro a esse circuito, o que permitirá que altere a velocidade das luzes enquanto o código está em execução.

Componentes necessários

Todos os componentes do projeto 5, mais...

Potenciômetro giratório de 4,7 Ω *



* Imagem cortesia de Iain Fergusson.

Conectando os componentes

Primeiramente, certifique-se de que o Arduino esteja desligado, desconectando-o do cabo USB. Agora, adicione o potenciômetro ao circuito como mostra a figura 3.2, com o terminal esquerdo indo para os 5 V do Arduino, o do meio para o pino analógico 2 e o da direita para o terra.

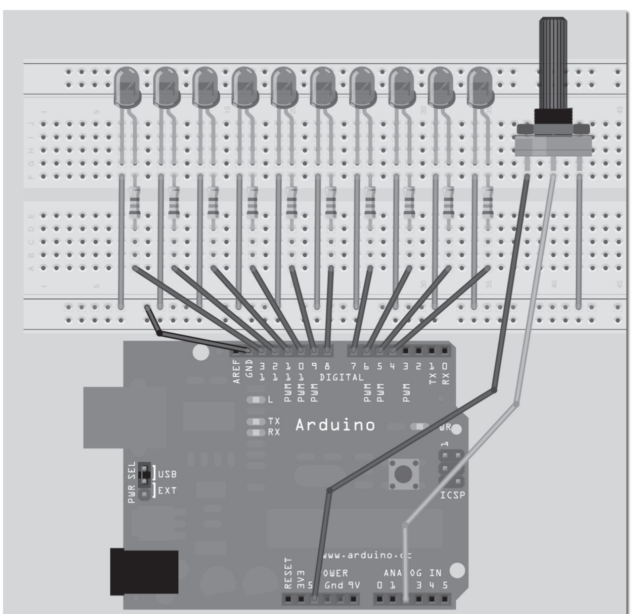


Figura 3.2 – Circuito do Projeto 6 – Efeito interativo de iluminação sequencial com LEDs (consulte o site da Novatec para versão colorida).

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 3.2.

Listagem 3.2 – Código para o projeto 6

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};    // Cria um array para os pinos dos LEDs
int ledDelay;     // intervalo entre as alterações
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
int potPin = 2;     // seleciona o pino de entrada para o potenciômetro

void setup() {
for (int x=0; x<10; x++) {     // define todos os pinos como saída
  pinMode(ledPin[x], OUTPUT); }
  changeTime = millis();
}

void loop() {
  ledDelay = analogRead(potPin);             // lê o valor do potenciômetro
  if ((millis() - changeTime) > ledDelay) {    // verifica se transcorreram ledDelay ms desde a
                                             // última alteração
    changeLED();
    changeTime = millis();
  }
}

void changeLED() {
  for (int x=0; x<10; x++) {   // apaga todos os LEDs
    digitalWrite(ledPin[x], LOW);
  }
  digitalWrite(ledPin[currentLED], HIGH);   // acende o LED atual
  currentLED += direction;           // incrementa de acordo com o valor de direction
  // altera a direção se tivermos atingido o fim
  if (currentLED == 9) {direction = -1;}
  if (currentLED == 0) {direction = 1;}
}
```

Depois de verificar e fazer o upload de seu código, você deverá ver os LEDs acenderem, indo e voltando de uma ponta a outra da sequência de luzes, como antes. Entretanto, virando o botão do potenciômetro, você pode alterar o valor de `ledDelay`, acelerando ou desacelerando o efeito.

Vamos, agora, observar como isso é feito, e descobrir o que é um potenciômetro.

Projeto 6 – Efeito interativo de iluminação sequencial com LEDs – Análise do código

O código deste projeto é praticamente idêntico ao do projeto anterior. Você simplesmente adicionou um potenciômetro ao seu hardware, e acrescentou o código necessário para permitir a leitura de valores do potenciômetro, e o uso desses valores, para ajustar a velocidade do efeito de iluminação sequencial dos LEDs.

Primeiramente, você declara uma variável para o pino do potenciômetro,

```
int potPin = 2;
```

Isso é feito dessa forma, pois seu potenciômetro está conectado ao pino analógico 2. Para ler o valor de um pino analógico, você utiliza o comando `analogRead`. O Arduino tem seis entradas/saídas analógicas com um conversor analógico-para-digital de 10 bits (discutiremos *bits* futuramente). Isso significa que o pino analógico pode ler voltagens, entre 0 e 5 volts, usando valores inteiros entre 0 (0 V) e 1.023 (5 V). Isso representa uma resolução de 5 V / 1024 unidades, ou 0,0049 V (4,9 mV) por unidade.

Defina seu intervalo de espera utilizando o potenciômetro, para que você possa usar os valores lidos diretamente do pino, ajustando o intervalo entre 0 e 1.023 milissegundos (ou pouco mais de um segundo). Isso é feito armazenando diretamente o valor do pino do potenciômetro em `ledDelay`. Note que você não tem de definir um pino analógico como entrada ou saída (diferentemente de um pino digital):

```
ledDelay = analogRead(potPin);
```

Isso ocorre durante seu loop principal e, portanto, é um valor que está constantemente sendo lido e ajustado. Virando o botão, você pode ajustar o valor do intervalo entre 0 e 1.023 milissegundos (ou pouco mais de um segundo), com controle total sobre a velocidade do efeito.

Agora, vamos descobrir o que é um potenciômetro e como ele funciona.

Projeto 6 – Efeito interativo de iluminação sequencial com LEDs – Análise do hardware

O único componente adicional de hardware utilizado neste projeto é o potenciômetro de 4K7 (4700 Ω).

Você sabe como resistores funcionam. O potenciômetro é apenas um resistor ajustável, com um alcance de zero a um valor definido (escrito em sua lateral). Neste projeto, você está utilizando um potenciômetro de 4K7 (4700 Ω), o que significa que seu alcance é de 0 Ω a 4700 Ω.

O potenciômetro tem três terminais. Conectando apenas dois, ele se torna um resistor variável. Conectando os três e aplicando uma voltagem, o potenciômetro se torna um divisor de tensão. É dessa forma que ele será utilizado em seu circuito. Um lado é conectado ao terra, outro aos 5 V, e o terminal central ao seu pino analógico.

Ajustando o botão, uma voltagem entre 0 V e 5 V será aplicada ao pino central; você pode ler o valor dessa voltagem no pino analógico 2 e utilizá-lo para alterar a taxa de intervalo empregada no efeito das luzes.

O potenciômetro pode ser muito útil, pois fornece uma forma de ajustar um valor entre zero e um limite definido, por exemplo, para regular o volume de um rádio ou o brilho de uma lâmpada. De fato, reostatos que regulam a luminosidade das lâmpadas de sua casa são um tipo de potenciômetro.

EXERCÍCIOS

Você tem todo o conhecimento necessário para ajustar o código e tentar os seguintes exercícios:

- **Exercício 1**: Faça com que os LEDs em AMBAS as pontas da sequência iniciem acesos. Então, avance as luzes, uma em direção da outra, dando a impressão de que estão se encontrando e retornando às pontas.
- **Exercício 2**: Faça um efeito de bola quicando, acendendo os LEDs na vertical. Acenda o LED da base e depois "quique", indo até o LED do topo, de volta à base, até o nono LED, de volta à base, até o oitavo LED e assim por diante, para simular uma bola quicando que perde momento a cada quique.

Projeto 7 – Lâmpada pulsante

Você agora tentará um método mais avançado de controle de seus LEDs. Até aqui você simplesmente os acendeu e apagou. Não seria interessante ajustar o brilho de um LED? Será que isso pode ser feito com o Arduino? Certamente.

Hora de retornar aos fundamentos.

Componentes necessários

LED verde difuso de 5 mm 

Resistores limitadores de corrente 

Conectando os componentes

O circuito para este projeto apresenta simplesmente um LED verde conectando, por meio de um resistor limitador de corrente, o terra e o pino digital 11 (Figura 3.3).

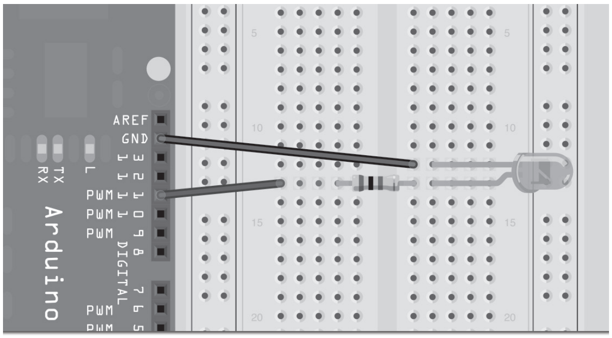


Figura 3.3 – Circuito para o Projeto 7 – Lâmpada pulsante (consulte o site da Novatec para versão colorida).

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 3.3.

Listagem 3.3 – Código para o projeto 7

```
// Projeto 7 - Lâmpada pulsante
int ledPin = 11;
float sinVal;
int ledVal;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  for (int x=0; x<180; x++) {
    // converte graus para radianos e, então, obtém o valor do seno
    sinVal = (sin(x*(3.1416/180)));
    ledVal = int(sinVal*255);
    analogWrite(ledPin, ledVal);
    delay(25);
  }
}
```

Verifique e faça o upload de seu código. Você verá seu LED pulsar, acendendo e apagando continuamente. Entretanto, em vez de um simples estado de ligado/desligado, você está ajustando o brilho. Vamos descobrir como isso funciona.

Projeto 7 – Lâmpada pulsante – Análise do código

O código para este projeto é muito simples, mas exige explicação.

Primeiramente, você define as variáveis para o pino do LED, um `float` (tipo de dado de ponto flutuante) para o valor da onda senoidal, e `ledVal`, que armazena o valor inteiro a ser enviado para o pino digital PWM 11.

Neste projeto, você está criando uma onda senoidal e fazendo com que o brilho do LED acompanhe essa onda. É isso que faz com que a luz pulse, em vez de simplesmente acender com brilho máximo e apagar novamente.

Você utiliza a função `sin()`, uma função matemática, para calcular o seno de um ângulo. Você tem de dar à função o grau em radianos. Assim, temos um loop `for` que vai de `0` a `179`; não devemos passar desse limite, pois isso resultaria em valores negativos, e o valor do brilho pode estar apenas entre 0 e 255.

A função `sin()` requer que o ângulo esteja em radianos, e não em graus. Assim, a equação `x*(3.1416/180)` converterá o grau do ângulo para radianos. Depois, você transfere o resultado para `ledVal`, multiplicando-o por 255 para obter seu valor. O resultado da função `sin()` será um número entre -1 e 1, que poderá ser multiplicado por 255 para obter o brilho máximo. Você *converte* (*cast*) o valor de tipo ponto flutuante de `sinVal` para um inteiro, utilizando `int()` na instrução a seguir:

```
ledVal = int(sinVal*255);
```

Então, você envia esse valor para o pino digital PWM 11, utilizando a instrução:

```
analogWrite(ledPin, ledVal);
```

Na conversão, ou *casting*, você converte o valor de ponto flutuante para um inteiro (efetivamente eliminando o que havia após o ponto decimal). Mas como você pode enviar um valor analógico para um pino digital? Dê uma olhada em seu Arduino. Se analisar os pinos digitais, você verá que seis deles (3, 5, 6, 9, 10 e 11) têm PWM escrito ao lado. Esses pinos são diferentes dos pinos digitais, pois são capazes de enviar um sinal PWM.

PWM significa Pulse Width Modulation, ou Modulação por Largura de Pulso (MLP), e representa uma técnica para obter resultados analógicos por meios digitais. Nesses pinos, o Arduino envia uma onda quadrada, ligando e desligando o pino muito rapidamente. O padrão de ligado/desligado pode simular uma voltagem variando entre 0 V e 5 V. Isso é feito alterando a quantidade de tempo em que a saída permanece alta (ligada) e baixa (desligada). A duração do tempo em que ela permanece ligada é conhecida como a *largura do pulso*.

Por exemplo, se você quisesse enviar o valor `0` para o pino digital PWM 11 utilizando `analogWrite()`, o período ON (ligado) seria zero, ou teria o que chamamos de um ciclo de trabalho (duty cycle[1]) de 0%. Se você quisesse enviar um valor de `63` (25% dos 255

1 N.T.: Em telecomunicações e eletrônica, o termo duty cycle (razão cíclica ou ciclo de trabalho) é utilizado para descrever a fração de tempo em que um sistema está em um estado "ativo" (fonte: Wikipédia).

máximos), o pino estaria ON em 25% do tempo e OFF em 75% do tempo. O valor de `191` teria um ciclo de trabalho de 75%; enquanto um valor de 255 teria um ciclo de trabalho de 100%. Os pulsos ocorrem a uma velocidade de aproximadamente 500 Hz, ou 2 milissegundos, cada.

Assim, em seu sketch, o LED está acendendo e apagando muito rapidamente. Se o ciclo de trabalho fosse de 50% (um valor de `127`), o LED pulsaria, acendendo e apagando, a 500 Hz, e exibiria metade de seu brilho máximo. Isso é basicamente uma ilusão que você pode utilizar em seu proveito, permitindo que os pinos digitais emitam um valor analógico simulado para seus LEDs.

Note que mesmo que apenas seis dos pinos tenham a função PWM, você pode facilmente escrever um software para conseguir uma saída PWM de todos os pinos digitais, se assim preferir.

Futuramente, revisitaremos o conceito de PWM para criar tons audíveis, utilizando um receptor acústico piezo.

Projeto 8 – Mood lamp[2] RGB

No projeto anterior, você aprendeu como ajustar o brilho de um LED utilizando as capacidades de PWM do chip Atmega. Agora, você aproveitará esse recurso, utilizando um LED vermelho, um verde e um azul, e misturará essas cores para criar a cor que quiser. Com isso, você criará uma mood lamp, como a que vemos em muitas lojas atualmente.

Componentes necessários

Dessa vez você utilizará três LEDs: um vermelho, um verde e um azul.

LED vermelho difuso de 5 mm


LED verde difuso de 5 mm


LED azul difuso de 5 mm


3 resistores limitadores de corrente


2 N.T.: Uma mood lamp é uma lâmpada de propósito geralmente estético, que serve para iluminar um ambiente mais com o intuito de indicar e caracterizar o "humor" (*mood*) do ambiente, do que propriamente servir como fonte de iluminação.

Conectando os componentes

Conecte os três LEDs como na figura 3.4. Pegue um pedaço de papel tamanho carta, enrole-o como um cilindro e prenda-o com uma fita. Coloque o cilindro sobre os três LEDs. Isso difundirá a luz de cada LED, mesclando as cores em uma única.

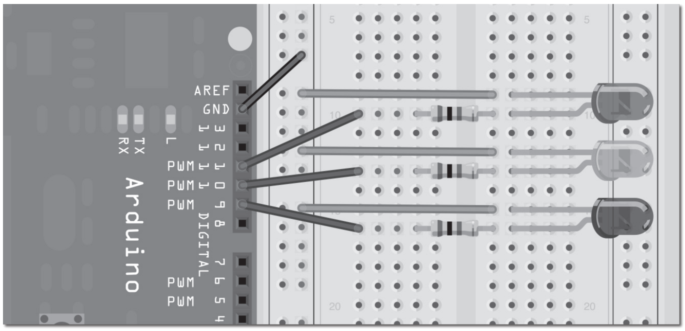


Figura 3.4 – Circuito para o Projeto 8 – Mood lamp RGB (consulte o site da Novatec para versão colorida).

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 3.4.

Listagem 3.4 – Código para o projeto 8

```
// Projeto 8 - Lâmpada de humor
float RGB1[3];
float RGB2[3];
float INC[3];

int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;

void setup() {
  randomSeed(analogRead(0));

  RGB1[0] = 0;
  RGB1[1] = 0;
  RGB1[2] = 0;

  RGB2[0] = random(256);
  RGB2[1] = random(256);
  RGB2[2] = random(256);
}
```

```
void loop() {
  randomSeed(analogRead(0));

  for (int x=0; x<3; x++) {
    INC[x] = (RGB1[x] - RGB2[x]) / 256; }

  for (int x=0; x<256; x++) {
    red = int(RGB1[0]);
    green = int(RGB1[1]);
    blue = int(RGB1[2]);

    analogWrite (RedPin, red);
    analogWrite (GreenPin, green);
    analogWrite (BluePin, blue);
    delay(100);

    RGB1[0] -= INC[0];
    RGB1[1] -= INC[1];
    RGB1[2] -= INC[2];
  }
  for (int x=0; x<3; x++) {
    RGB2[x] = random(556)-300;
    RGB2[x] = constrain(RGB2[x], 0, 255);
    delay(1000);
  }
}
```

Quando executar esse código, você verá as cores se alterando lentamente. Parabéns, você acabou de criar sua própria mood lamp!

Projeto 8 – Mood lamp RGB – Análise do código

A mood lamp é formada de três LEDs, um vermelho, um verde e um azul. Da mesma forma que seu monitor de computador é formado de pequenos pontos vermelhos, verdes e azuis (RGB), você pode gerar cores diferentes ajustando o brilho de cada um dos três LEDs, para que resultem em um valor RGB diferente.

Alternativamente, você poderia utilizar um LED RGB, um único LED de 5 mm com quatro terminais (alguns têm mais), no qual um desses terminais é um ânodo (positivo) ou cátodo (negativo) comum, e os outros três vão para o terminal oposto dos LEDs vermelho, verde e azul, dentro da lâmpada. Basicamente, são três LEDs coloridos espremidos em um único LED de 5 mm. Tais dispositivos são mais compactos, mas, também, mais caros.

Um valor RGB (255, 0, 0) é vermelho puro, enquanto um valor (0, 255, 0) é verde puro e (0, 0, 255), azul puro. Misturando esses valores você pode obter todas as cores. Esse é o modelo de cores aditivo (Figura 3.5). Repare que, mesmo que você estivesse simplesmente acendendo e apagando os LEDs (ou seja, sem utilizar brilhos diferentes), ainda conseguiria cores diferentes como resultado.

Tabela 3.1 – Cores disponíveis acendendo e apagando os LEDs em combinações diferentes

Vermelho	Verde	Azul	Cor
255	0	0	Vermelho
0	255	0	Verde
0	0	255	Azul
255	255	0	Amarelo
0	255	255	Ciano
255	0	255	Magenta
255	255	255	Branco

Difundindo a luz com o cilindro de papel, você mistura as cores agradavelmente. Os LEDs podem ser colocados dentro de qualquer objeto capaz de difundir a luz; outra opção é refletir a luz com um difusor reflexivo. Experimente colocar as luzes dentro de uma bola de pingue-pongue, ou em uma pequena garrafa plástica (quanto mais fino o plástico, melhor).

Ao ajustar o brilho utilizando PWM, você também pode obter outras cores. Colocando os LEDs próximos uns dos outros e misturando seus valores, o espectro de luz das três cores adicionadas cria uma única cor (Figura 3.5). O número total de cores disponíveis, utilizando PWM com um intervalo de 0 a 255, é de 16.777.216 cores (256 x 256 x 256).

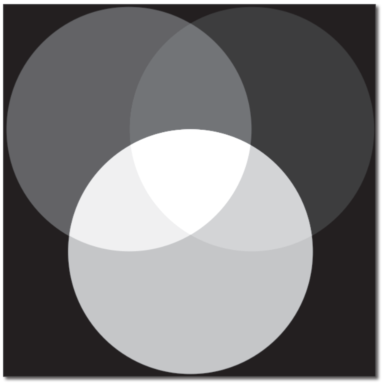


Figura 3.5 – Misturando R,G e B para obter cores diferentes (consulte o site da Novatec para versão colorida).

No código, você inicia declarando alguns arrays de ponto flutuante e variáveis de valor inteiro para armazenar seus valores RGB, assim como um valor de incremento, da seguinte maneira:

```
float RGB1[3];
float RGB2[3];
float INC[3];

int red, green, blue;
```

Na função de inicialização (*setup*), você tem o seguinte:

```
randomSeed(analogRead(0));
```

O comando `randomSeed` cria números aleatórios (na verdade, pseudoaleatórios). Um chip de computador não é capaz de produzir números verdadeiramente aleatórios, por isso ele procura dados em uma parte de sua memória que possam ser diferentes, ou uma tabela de valores diferentes, e utiliza esses dados como números pseudoaleatórios. Definindo uma *semente* (*seed*), você pode dizer ao computador onde na memória, ou em qual tabela, ele deve iniciar a contagem. Nesse caso, o valor atribuído a `randomSeed` é um valor lido do pino analógico 0. Como não há nada conectado a ele, será lido um número aleatório, criado por ruído analógico.

Assim que você tiver definido uma semente, poderá criar um número aleatório utilizando a função `random()`. Com isso, você terá dois conjuntos de valores RGB armazenados em arrays de três elementos. `RGB1` contém os valores RGB iniciais da lâmpada (nesse caso, todos são zero, ou desligados):

```
RGB1[0] = 0;
RGB1[1] = 0;
RGB1[2] = 0;
```

Enquanto isso, o array `RGB2` é um conjunto de valores RGB aleatórios, para os quais você deseja que a lâmpada faça sua transição:

```
RGB2[0] = random(256);
RGB2[1] = random(256);
RGB2[2] = random(256);
```

Nesse caso, você definiu os valores como um número aleatório, definido por `random(256)`, o que resultará em um número entre 0 e 255 (o limite superior não entra no intervalo de resultados da função porque o número sempre irá variar de zero em diante).

Se você passar um único número à função `random()`, ela retornará um valor entre zero e o número passado menos um, por exemplo, `random(1000)` retornará um número entre 0 e 999. Se você fornecer dois números como parâmetros, ela retornará um número aleatório entre o número menor (incluindo ele próprio) e o número maior (menos um), por exemplo, `random(10,100)` retornará um número aleatório entre 10 e 99.

No loop principal do programa, primeiro você consulta os valores RGB de início e de fim, e avalia qual valor é necessário como incremento, para que possamos avançar de um valor para o outro em 256 passos (uma vez que o valor PWM pode estar apenas entre 0 e 255). Isso é feito da seguinte maneira:

```
for (int x=0; x<3; x++) {
    INC[x] = (RGB1[x] - RGB2[x]) / 256; }
```

Esse loop `for` define os valores de incremento para os canais R, G e B, calculando a diferença entre os dois valores de brilho e dividindo o resultado por `256`.

Você também tem outro loop `for`

```
for (int x=0; x<256; x++) {
  red = int(RGB1[0]);
  green = int(RGB1[1]);
  blue = int(RGB1[2]);

  analogWrite (RedPin, red);
  analogWrite (GreenPin, green);
  analogWrite (BluePin, blue);
  delay(100);

  RGB1[0] -= INC[0];
  RGB1[1] -= INC[1];
  RGB1[2] -= INC[2];
}
```

Esse loop primeiro pega os valores de vermelho, verde e azul do array RGB1; depois, ele escreve esses valores nos pinos digitais 9, 10 e 11, subtrai o valor de incremento e repete esse processo 256 vezes para avançar de uma cor a outra. A espera de 100ms entre cada passo garante uma progressão lenta e constante. Você pode, evidentemente, ajustar esse valor, se quiser que as alterações ocorram mais rapidamente, ou mais lentamente; também podemos adicionar um potenciômetro, para permitir que o usuário regule a velocidade.

Depois de percorrer 256 passos entre uma cor aleatória e a próxima, o array `RGB1` terá praticamente os mesmos valores do array `RGB2`. Você, agora, deverá decidir por outro conjunto de valores aleatórios para a próxima transição. Isso é feito com outro loop `for`:

```
for (int x=0; x<3; x++) {
  RGB2[x] = random(556)-300;
  RGB2[x] = constrain(RGB2[x], 0, 255);
  delay(1000);
}
```

O número aleatório é escolhido selecionando um número entre 0 e 555 (255 + 300) e subtraindo 300. Dessa maneira, você está tentando forçar a ocorrência, de tempos em tempos, das cores primárias, para garantir que não tenha sempre apenas tons pastéis.

Você tem 300 chances em 556 de obter um número negativo, e de criar, dessa forma, uma tendência a um ou mais dos outros dois canais de cores. O comando seguinte garante que os números enviados aos pinos PWM não sejam negativos, utilizando a função `constrain()`.

A função `constrain()` requer três parâmetros: `x`, `a` e `b`, em que `x` é o número que você deseja restringir, `a` é o menor valor do intervalo e `b`, o maior valor. Assim, a função `constrain()` verifica o valor de `x` e garante que ele esteja entre `a` e `b`. Se `x` for menor que `a`, a função o define como `a`; se `x` for maior que `b`, ela o define como `b`. Em seu caso, você assegura que o número esteja entre 0 e 255, intervalo de sua saída PWM.

Como você utiliza `random(556)-300` para seus valores RGB, alguns desses valores serão menores que zero; a função de restrição garante que o valor enviado ao PWM não seja menor que zero.

EXERCÍCIO

Veja se você consegue alterar o código para utilizar as cores do arco-íris, em vez de cores aleatórias.

Projeto 9 – Efeito de fogo com LEDs

O projeto 9 utilizará LEDs e um efeito aleatório de cintilação, novamente via PWM, para imitar o efeito de uma chama cintilante. Se você colocar esses LEDs dentro de uma pequena casa ou de um modelo de ferromodelismo, poderá fazer com que pareça que a casa está pegando fogo. Você também pode utilizar esse recurso na lareira de sua casa, em vez de toras de madeira. Esse é um exemplo simples de como LEDs podem ser utilizados para criar efeitos especiais para filmes, peças de teatro, dioramas, modelos de ferromodelismo etc.

Componentes necessários

Desta vez, você utilizará três LEDs: um vermelho e dois amarelos.

LED vermelho difuso de 5 mm



2 LEDs amarelos difusos de 5 mm



3 resistores limitadores de corrente



Conectando os componentes

Desligue seu Arduino, depois conecte seus LEDs como na Figura 3.6. Esse é essencialmente o mesmo circuito do projeto 8, mas utiliza um LED vermelho e dois amarelos, em vez de um vermelho, um verde e um azul. Uma vez mais, o efeito é mais bem observado quando a luz é difundida utilizando um cilindro de papel, ou quando refletida em um papel branco, ou espelhada em outra superfície.

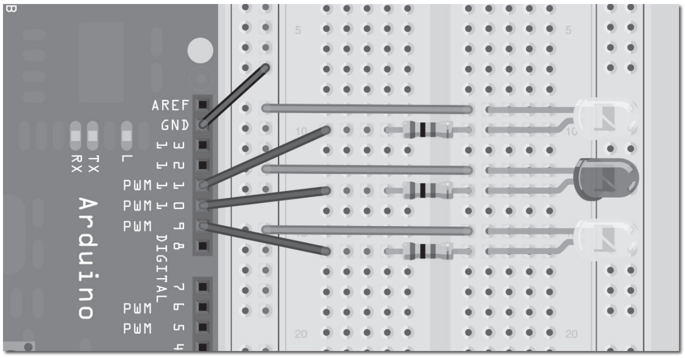


Figura 3.6 – Circuito para o Projeto 9 – Efeito de fogo com LEDs (consulte o site da Novatec para versão colorida).

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 3.5.

Listagem 3.5 – Código para o projeto 9

```
// Projeto 9 - Efeito de fogo com LEDs
int ledPin1 = 9;
int ledPin2 = 10;
int ledPin3 = 11;

void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}

void loop() {
  analogWrite(ledPin1, random(120)+136);
  analogWrite(ledPin2, random(120)+136);
  analogWrite(ledPin3, random(120)+136);
  delay(random(100));
}
```

Pressione o botão **Verify/Compile** no topo do IDE para garantir que não haja erros em seu código. Se não houver erros, clique no botão **Upload**.

Se tudo foi feito corretamente, os LEDs passarão a cintilar de modo aleatório, simulando uma chama ou um efeito de fogo.

Projeto 9 – Efeito de fogo com LEDs – Análise do código

Vamos analisar o código para este projeto. Primeiramente, você declara e inicializa algumas variáveis de tipo inteiro, para armazenar os valores dos pinos digitais aos quais você conectará seus LEDs:

```
int ledPin1 = 9;
int ledPin2 = 10;
int ledPin3 = 11;
```

Depois, você os define como saídas:

```
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
pinMode(ledPin3, OUTPUT);
```

O loop principal do programa envia um valor aleatório entre 0 e 119; adicione 136 a ele para obter o brilho máximo do LED para os pinos PWM 9, 10 e 11:

```
analogWrite(ledPin1, random(120)+136);
analogWrite(ledPin2, random(120)+136);
analogWrite(ledPin3, random(120)+136);
```

Por fim, você tem uma espera aleatória entre 0 e 99ms:

```
delay(random(100));
```

Então o loop principal reinicia, provocando o efeito de cintilação. Reflita a luz em um papel branco, ou espelhe o efeito em sua parede, e você terá um efeito realístico de fogo.

O hardware é simples e você deve conseguir compreendê-lo sem problemas, por isso vamos avançar para o projeto 10.

EXERCÍCIOS

Experimente estes dois exercícios:

- **Exercício 1**: Utilizando um ou dois LEDs azuis e/ou brancos, veja se você consegue recriar o efeito dos raios de luz de um soldador a arco.
- **Exercício 2**: Utilizando LEDs azuis e vermelhos, recrie os efeitos de luz de um veículo de emergência.

Projeto 10 – Mood lamp com controle serial

Para o projeto 10, você revisitará o circuito do projeto 8 (o da mood lamp RGB), mas agora mergulhará no mundo das comunicações seriais. Você controlará sua lâmpada enviando comandos do PC para o Arduino utilizando o Serial Monitor, do IDE do Arduino. A comunicação serial é o processo de envio de dados, um bit de cada vez, por um link de comunicação.

Este projeto também apresenta como manipular strings de texto. Portanto, prepare seu hardware da mesma forma que no projeto 8, e digite o novo código.

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 3.6.

Listagem 3.6 – Código para o projeto 10

```
// Projeto 10 - Mood lamp com controle serial
char buffer[18];
int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;

void setup() {
  Serial.begin(9600);
  Serial.flush();
  pinMode(RedPin, OUTPUT);
  pinMode(GreenPin, OUTPUT);
  pinMode(BluePin, OUTPUT);
}

void loop() {
  if (Serial.available() > 0) {
    int index=0;
    delay(100); // deixe o buffer encher
    int numChar = Serial.available();
    if (numChar>15) {
      numChar=15;
     }
    while (numChar--) {
      buffer[index++] = Serial.read();
    }
    splitString(buffer);
  }
}
```

```
void splitString(char* data) {
  Serial.print("Data entered: ");
  Serial.println(data);
  char* parameter;
  parameter = strtok (data, " ,");
  while (parameter != NULL) {
    setLED(parameter);
    parameter = strtok (NULL, " ,");
}

   // Limpa o texto e os buffers seriais
  for (int x=0; x<16; x++) {
    buffer[x]='\0';
   }
  Serial.flush();
}

void setLED(char* data) {
  if ((data[0] == 'r') || (data[0] == 'R')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(RedPin, Ans);
    Serial.print("Red is set to: ");
    Serial.println(Ans);
  }
  if ((data[0] == 'g') || (data[0] == 'G')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(GreenPin, Ans);
    Serial.print("Green is set to: ");
    Serial.println(Ans);
  }
  if ((data[0] == 'b') || (data[0] == 'B')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(BluePin, Ans);
    Serial.print("Blue is set to: ");
    Serial.println(Ans);
  }
}
```

Assim que você tiver verificado o código, faça seu upload para o Arduino.

Note que, quando você faz o upload do programa, nada parece acontecer. Isso ocorre porque o programa está aguardando por uma entrada sua. Inicie o Serial Monitor, clicando em seu ícone na barra de tarefas do IDE do Arduino.

Na janela de texto do monitor serial, digite manualmente os valores R, G e B para cada um dos três LEDs. Os LEDs mudarão para as cores que você determinou.

Se você digitar `R255`, o LED vermelho acenderá com brilho máximo. Se você digitar `R255, G255`, tanto o LED vermelho quanto o verde acenderão com brilho máximo. Agora, digite `R127, G100, B255`, e você terá uma agradável cor púrpura. Se digitar `r0, g0, b0`, você apagará todos os LEDs.

O texto de entrada é projetado para aceitar as letras R, G e B em maiúsculas ou minúsculas, seguidas por um valor de 0 a 255. Qualquer valor acima de 255 será limitado a 255 por padrão. Você pode digitar uma vírgula ou espaço entre os parâmetros, e pode digitar um, dois ou três valores de LED a qualquer momento; por exemplo:

```
r255 b100

r127 b127 g127

G255, B0

B127, R0, G255
```

Projeto 10 – Mood lamp com controle serial – Análise do código

Este projeto apresenta vários novos conceitos, incluindo comunicação serial, ponteiros e manipulação de strings.

Primeiramente, você define um array de 18 caracteres (`char`) para armazenar sua string de texto, que é maior do que o máximo de 16 permitidos, para garantir que você não tenha erros de estouro de buffer.

```
char buffer[18];
```

Em seguida, você define os inteiros para armazenar os valores `red`, `green` e `blue`, assim como os valores para os pinos digitais:

```
int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;
```

Em sua função de inicialização, você define os três pinos digitais como saídas. Antes disso, porém, temos o comando `Serial.begin`:

```
void setup() {
  Serial.begin(9600);
  Serial.flush();
  pinMode(RedPin, OUTPUT);
  pinMode(GreenPin, OUTPUT);
  pinMode(BluePin, OUTPUT);
}
```

`Serial.begin` diz ao Arduino para iniciar comunicações seriais; o número dentro dos parênteses (nesse caso, `9600`) define a taxa de transmissão (de símbolos ou pulsos por segundo) na qual a linha serial se comunicará.

O comando `Serial.flush` liberará caracteres que estejam na linha serial, deixando-a vazia e pronta para entradas/saídas.

A linha de comunicações seriais é simplesmente uma forma do Arduino se comunicar com o mundo externo, que, nesse caso, ocorre de, e para, o PC e o Serial Monitor do IDE do Arduino.

No loop principal você tem uma instrução `if`:

```
if (Serial.available() > 0) {
```

Essa instrução está utilizando o comando `Serial.available` para verificar se algum caractere foi enviado pela linha serial. Se caracteres já tiverem sido recebidos, a condição é atendida e o código dentro do bloco de instruções `if` é executado:

```
if (Serial.available() > 0) {
    int index=0;
  delay(100); // deixe o buffer encher
  int numChar = Serial.available();
  if (numChar>15) {
    numChar=15;
  }
  while (numChar--) {
    buffer[index++] = Serial.read();
  }
  splitString(buffer);
}
```

Um inteiro, `index`, é declarado e inicializado como zero. Esse inteiro armazenará a posição de um ponteiro para os caracteres do array `char`.

Você então define uma espera de `100`, para garantir que o buffer serial (local na memória em que os dados recebidos são armazenados, antes do processamento) esteja cheio antes de processar os dados. Caso você não faça isso, é possível que a função seja executada e inicie o processamento da string de texto antes de você receber todos os dados. A linha de comunicações seriais é muito lenta quando comparada à velocidade de execução do restante do código. Quando você envia uma string de caracteres, a função `Serial.available` terá imediatamente um valor maior que zero, e a função `if` iniciará sua execução. Se você não utilizasse a instrução `delay(100)`, ela poderia iniciar a execução do código da instrução `if` antes que toda a string tivesse sido recebida, e os dados seriais poderiam ser apenas os primeiros caracteres da string de texto.

Depois de esperar 100ms para que o buffer serial se encha com os dados enviados, você declara e inicializa o inteiro `numChar` com o número de caracteres da string de texto.

Assim, se enviássemos este texto no Serial Monitor

```
R255, G255, B255
```

O valor de `numChar` seria `17`. Ele seria `17` e não `16`, pois ao final de cada linha de texto há um caractere invisível, o caractere `NULL`, que informa ao Arduino quando atingimos o final da linha de texto.

A próxima instrução `if` verifica se o valor de `numChar` é maior que `15`; se for, ela o define como `15`. Isso garante que você não estoure o array `char buffer[18]`.

Em seguida, temos um comando `while`, algo que você ainda não viu, por isso deixe-me explicá-lo.

Sua sintaxe é a seguinte:

```
while(expression) {
  // instrução(ões)
}
```

Em seu código, o loop `while` é:

```
while (numChar--) {
    buffer[index++] = Serial.read();
  }
```

A condição que ele está verificando é `numChar`. Em outras palavras, o loop verifica se o valor armazenado no inteiro `numChar` não é zero. Note que `numChar` tem `--` depois dele. Isso é um pós-decremento: o valor é diminuído *depois* de utilizado. Se você tivesse utilizado `--numChar`, o valor em `numChar` seria diminuído (subtraído em 1) antes de ser avaliado. Em nosso caso, o loop `while` verifica o valor de `numChar` e, apenas depois, subtrai 1 dele. Se o valor de `numChar` for diferente de zero antes do decremento, o código do bloco será executado.

`numChar` está definida como o comprimento da string de texto que você digitou na janela do Serial Monitor. Assim, o código do loop `while` executará várias vezes.

O código do loop `while` é:

```
buffer[index++] = Serial.read();
```

Isso define cada elemento do array `buffer` como os caracteres lidos a partir da linha serial. Em outras palavras, preenche o array `buffer` com as letras que você digitou na janela de texto do Serial Monitor.

O comando `Serial.read()` lê dados seriais, um bit por vez. Dessa forma, agora que seu array de caracteres foi preenchido com os caracteres que você digitou no monitor serial, o loop `while` terminará assim que `numChar` atingir zero (por exemplo, o comprimento da string).

Depois do loop `while`, temos

```
splitString(buffer);
```

Uma chamada a uma das duas funções que você criou com o nome de `splitString()`. A função se parece com isso:

```
void splitString(char* data) {
  Serial.print("Data entered: ");
  Serial.println(data);
  char* parameter;
  parameter = strtok (data, " ,");
  while (parameter != NULL) {
    setLED(parameter);
    parameter = strtok (NULL, " ,");
  }
  // Limpa o texto e os buffers seriais
  for (int x=0; x<16; x++) {
    buffer[x]='\0';
  }
  Serial.flush();
}
```

Como ela não retorna nenhum dado, seu tipo de dados foi definido como `void`. Você passa a ela um parâmetro: uma variável do tipo `char`, que você chama de `data`. Entretanto, nas linguagens de programação C e C++, não é permitido que você envie um array de caracteres a uma função. Você contorna essa limitação utilizando um ponteiro. Você sabe que está utilizando um ponteiro, pois um asterisco foi adicionado ao nome da variável `*data`.

Ponteiros representam um tópico avançado em C, por isso não entrarei em muitos detalhes sobre eles. Caso você queira saber mais, consulte um livro sobre programação em C. Tudo que você tem de saber por ora é que, declarando `data` como um ponteiro, ela se torna uma variável que aponta para outra variável. Você pode apontar para o endereço no qual a variável está armazenada na memória utilizando o símbolo `&`, ou, em nosso caso, apontar para o valor armazenado nesse endereço na memória utilizando o símbolo `*`. Você utiliza esse recurso para contornar uma limitação, uma vez que, como mencionamos, não é permitido que você envie um array de caracteres a uma função. Entretanto, você pode enviar à sua função um ponteiro que aponte para um array de caracteres. Assim, você declarou uma variável de tipo de dado `char`, e chamou-a de `data`, mas o símbolo `*` antes dela significa que ela está apontando para o valor armazenado na variável `buffer`.

Ao chamar `splitString()`, você envia o conteúdo de `buffer` (mais propriamente, um ponteiro para ele, como acabamos de ver):

```
splitString(buffer);
```

Assim, você chamou a função e passou a ela todo o conteúdo do array de caracteres `buffer`.

O primeiro comando é

```
Serial.print("Data entered: ");
```

Essa é sua forma de enviar dados do Arduino para o PC. Nesse caso, o comando `print` envia o que estiver dentro dos parênteses para o PC, por meio do cabo USB, e você pode ler o resultado na janela do Serial Monitor. Aqui, você enviou as palavras `"Data entered: "`. Note que o texto deve estar entre aspas. A próxima linha é semelhante:

```
Serial.println(data);
```

Novamente, você enviou dados de volta ao PC. Dessa vez, você enviou a variável `data`, que é uma cópia do conteúdo do array de caracteres `buffer` que você passou à função. Assim, se a string de texto digitada for

```
R255 G127 B56
```

Então o comando

```
Serial.println(data);
```

Enviará essa string de texto de volta ao PC, exibindo-a na janela do Serial Monitor. (Certifique-se de que você primeiro habilitou a janela do Serial Monitor.)

Dessa vez, o comando `print` tem `ln` ao seu final: `println`. Isso significa simplesmente "imprima com um avanço de linha".

Quando você imprime utilizando o comando `print`, o cursor (o ponto em que o próximo símbolo de texto surgirá) permanece ao final do que você imprimiu. Quando você utiliza o comando `println`, um comando de avanço de linha é emitido: imprime-se o texto e depois o cursor desce para a linha seguinte.

```
Serial.print("Data entered: ");
Serial.println(data);
```

Dessa forma, se você analisar os dois comandos `print`, verá que o primeiro imprime `"Data entered: "`, deixando o cursor ao final do texto, enquanto o comando de impressão seguinte imprime `data` (o conteúdo do array `buffer`) e emite um avanço de linha, descendo o cursor para a linha seguinte. Se, depois disso, você utilizar mais uma instrução `print` ou `println`, o texto no monitor serial surgirá na linha seguinte.

Agora, você cria uma nova variável de tipo `char`, `parameter`

```
char* parameter;
```

Como você está utilizando essa variável para acessar elementos do array `data`, ela deve ser do mesmo tipo, daí o símbolo `*`. Você não pode passar dados de uma variável de determinado tipo de dados para uma de outro tipo; os dados devem ser primeiramente convertidos. Essa variável é outro exemplo de variável que tem *escopo local* e que

pode ser vista apenas pelo código dentro dessa função. Assim, se você tentar acessar a variável `parameter` fora da função `splitString()`, receberá um erro.

Depois, você utiliza um comando `strtok`, muito útil para manipulação de strings de texto. `strtok` recebe esse nome por causa das palavras String e Token, pois tem como propósito dividir uma string utilizando tokens. Em seu caso, o token que estamos procurando é um espaço ou vírgula; o comando está sendo utilizado para dividir strings de texto em strings menores.

Você passa o array `data` ao comando `strtok` como primeiro argumento e os tokens (entre aspas) como segundo argumento. Portanto:

```
parameter = strtok (data, " ,");
```

Com isso, `strtok` divide a string quando encontra um espaço ou uma vírgula.

Se sua string de texto for

```
R127 G56 B98
```

Então, depois dessa instrução, o valor de `parameter` será

```
R127
```

Pois o comando `strtok` divide a string na primeira ocorrência de uma vírgula.

Depois que você tiver definido a variável `parameter` como a seção da string de texto que deseja extrair (por exemplo, o trecho até o primeiro espaço ou vírgula), você entra em um loop `while` com a condição de que `parameter` não esteja vazia (por exemplo, se você tiver atingido o final da string):

```
while (parameter != NULL) {
```

Dentro do loop chamamos nossa segunda função:

```
setLED(parameter);
```

(Veremos essa função mais detalhadamente no futuro.) Depois, você define a variável `parameter` como a próxima seção da string, até o próximo espaço ou vírgula. Isso é feito passando para `strtok` um parâmetro `NULL`, da seguinte maneira:

```
parameter = strtok (NULL, " ,");
```

Isso diz ao comando `strtok` para continuar de onde tinha parado.

Assim, todo este trecho da função:

```
char* parameter;
parameter = strtok (data, " ,");
while (parameter != NULL) {
  setLED(parameter);
  parameter = strtok (NULL, " ,");
  }
```

Está simplesmente removendo cada parte da string de texto, separadas por espaços ou vírgulas, e enviando essas partes à próxima função, `setLED()`.

O trecho final dessa função simplesmente preenche o array `buffer` com caracteres `NULL`, o que é feito com o símbolo `\0`, e então libera os dados do buffer serial, deixando-o pronto para a entrada do próximo conjunto de dados:

```
// Limpe o texto e os buffers seriais
for (int x=0; x<16; x++) {
  buffer[x]='\0';
}
Serial.flush();
```

A função `setLED()` tomará cada parte da string de texto e definirá o LED correspondente com a cor que você escolheu. Assim, se a string de texto digitada por você for

```
G125 B55
```

A função `splitString()` dividirá essa linha em dois componentes separados

```
G125
B55
```

E enviará essas strings de texto abreviadas para a função `setLED()`, que fará a leitura dos dados e decidirá qual LED você escolheu, definindo para ele o valor de brilho correspondente.

Vamos retornar à segunda função, `setLED()`:

```
void setLED(char* data) {
  if ((data[0] == 'r') || (data[0] == 'R')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(RedPin, Ans);
    Serial.print("Red is set to: ");
    Serial.println(Ans);
  }
  if ((data[0] == 'g') || (data[0] == 'G')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(GreenPin, Ans);
    Serial.print("Green is set to: ");
    Serial.println(Ans);
  }
  if ((data[0] == 'b') || (data[0] == 'B')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(BluePin, Ans);
    Serial.print("Blue is set to: ");
```

```
    Serial.println(Ans);
  }
}
```

Essa função contém três instruções `if` semelhantes, por isso vamos escolher uma e analisá-la:

```
if ((data[0] == 'r') || (data[0] == 'R')) {
  int Ans = strtol(data+1, NULL, 10);
  Ans = constrain(Ans,0,255);
  analogWrite(RedPin, Ans);
  Serial.print("Red is set to: ");
  Serial.println(Ans);
}
```

Essa instrução `if` verifica se o primeiro caractere na string `data[0]` é a letra r ou R (caracteres maiúsculos ou minúsculos são totalmente diferentes para a linguagem C). Você utiliza o comando lógico OU (o símbolo `||`) para verificar se a letra é um r *ou* um R, uma vez que as duas opções são válidas.

Se ela é um r ou um R, a instrução `if` sabe que você deseja alterar o brilho do LED vermelho (red), e seu código é executado. Primeiramente, você declara uma variável inteira `Ans` (que tem escopo local, apenas para a função `setLED`) e utiliza o comando `strtol` (string para inteiro longo), para converter os caracteres depois da letra R em um inteiro. O comando `strtol` aceita três parâmetros: a string que você está passando, um ponteiro para o caractere depois do inteiro (que você não utilizará, pois já limpou a string utilizando o comando `strtok` e, portanto, passa um caractere `NULL`), e a base (nesse caso, base 10, pois você está utilizando números decimais normais, em vez de números binários, octais ou hexadecimais, que seriam base 2, 8 e 16, respectivamente). Em resumo, você declara um inteiro e o define como o valor da string de texto depois da letra R (o trecho com o número).

Depois, você utiliza o comando `constrain` para garantir que `Ans` vá apenas de `0` a `255` e não passe disso. Então, você utiliza um comando `analogWrite` para o pino vermelho, enviando a ele o valor de `Ans`. O código emite `"Red is set to: "`, seguido pelo valor de `Ans`, no Serial Monitor. As outras duas instruções `if` fazem exatamente o mesmo procedimento, só que para os LEDs verde e azul.

Você encontrou muitos novos tópicos e conceitos neste projeto. Para garantir que compreendeu exatamente o que fizemos, coloquei o código do projeto (que está em C, lembre-se) lado a lado com o pseudocódigo correspondente (essencialmente, a linguagem do computador descrita com mais detalhes, utilizando palavras e pensamentos). Consulte a tabela 3.2 para a comparação.

Tabela 3.2 – Explicação para o código do projeto 10 utilizando pseudocódigo

Linguagem de programação C	Pseudocódigo
`// Projeto 10 – Lâmpada de humor com controle serial`	Comentário com o número e nome do projeto
`char buffer[18];`	Declare um array de caracteres de 18 letras
`int red, green, blue;`	Declare 3 inteiros: red, green e blue
`int RedPin = 11;`	Um inteiro atribuindo determinado pino ao LED vermelho
`int GreenPin = 10;`	Um inteiro atribuindo determinado pino ao LED verde
`int BluePin = 9;`	Um inteiro atribuindo determinado pino ao LED azul
`void setup()`	Função setup
`{`	
`  Serial.begin(9600);`	Defina as comunicações seriais em 9.600 caracteres por segundo
`  Serial.flush();`	Libere a linha serial
`  pinMode(RedPin, OUTPUT);`	Defina o pino vermelho como pino de saída
`  pinMode(GreenPin, OUTPUT);`	O mesmo para o verde
`  pinMode(BluePin, OUTPUT);`	E para o azul
`}`	
`void loop()`	O loop principal do programa
`{`	
`  if (Serial.available() > 0) {`	Se dados estão sendo enviados pela linha serial...
`    int index=0;`	Declare um inteiro, index, e o defina como 0
`    delay(100); // Deixe o buffer encher`	Espere 100 milissegundos
`    int numChar = Serial.available();`	Defina numChar como os dados entrantes vindos do serial
`    if (numChar>15) {`	Se numChar for maior que 15 caracteres...
`      numChar=15;`	Faça com que tenha no máximo 15 caracteres
`    }`	
`    while (numChar--) {`	Enquanto numChar não for 0 (subtraia 1 dele)
`      buffer[index++] = Serial.read();`	Defina element[index] como o valor lido (adicione 1)
`    }`	
`    splitString(buffer);`	Chame a função splitString e envie a ela dados do buffer
`  }`	
`}`	
`void splitString(char* data) {`	A função splitString faz referência aos dados do buffer
`  Serial.print("Data entered: ");`	Imprima "Data entered: "
`  Serial.println(data);`	Imprima o valor dos dados e depois desça uma linha
`  char* parameter;`	Declare parameter como o tipo de dado char
`  parameter = strtok (data, " ,");`	Defina-o como o texto até o primeiro espaço ou vírgula
`  while (parameter != NULL) {`	Enquanto o conteúdo de parameter não estiver vazio...
`    setLED(parameter);`	Chame a função SetLED
`    parameter = strtok (NULL, " ,");`	Defina parameter como a próxima parte da string de texto
`  }`	
`  // Limpe o texto e os buffers seriais`	Outro comentário
`    for (int x=0; x<16; x++) {`	Repita a linha seguinte 16 vezes
`      buffer[x]='\0';`	Defina cada elemento de buffer como NULL (vazio)
`  }`	
`  Serial.flush();`	Libere as comunicações seriais
`}`	
	Uma função setLED recebe o buffer
	Se a primeira letra for r ou R...
	Defina o inteiro Ans como o número na próxima parte do texto
	Certifique-se de que ele esteja entre 0 e 255
	Escreva esse valor no pino vermelho
	Imprima "Red is set to: "
	E então o valor de Ans

<table>
<tr>
<td><pre>void setLED(char* data) {
  if ((data[0] == 'r') || (data[0] == 'R')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(RedPin, Ans);
    Serial.print("Red is set to: ");
    Serial.println(Ans);
  }
  if ((data[0] == 'g') || (data[0] == 'G')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(GreenPin, Ans);
    Serial.print("Green is set to: ");
    Serial.println(Ans);
  }
  if ((data[0] == 'b') || (data[0] == 'B')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(BluePin, Ans);
    Serial.print("Blue is set to: ");
    Serial.println(Ans);
  }
}</pre></td>
<td>Se a primeira letra for g ou G...
Defina o inteiro Ans como o número na próxima parte do texto
Certifique-se de que ele esteja entre 0 e 255
Escreva esse valor no pino verde
Imprima "Green is set to: "
E então o valor de Ans

Se a primeira letra for b ou B...
Defina o inteiro Ans como o número na próxima parte do texto
Certifique-se de que ele esteja entre 0 e 255
Escreva esse valor no pino azul
Imprima "Blue is set to: "
E então o valor de Ans</td>
</tr>
</table>

Esperamos que o pseudocódigo mostrado ajude você a compreender exatamente o que ocorre no código.

Resumo

O capítulo 3 apresentou muitos comandos e conceitos de programação novos. Você aprendeu sobre arrays e como utilizá-los, como ler valores analógicos de um pino, como utilizar pinos PWM, e o básico sobre comunicações seriais. Saber como enviar e ler dados por meio de uma linha serial significa que você pode utilizar seu Arduino para se comunicar com todos os tipos de dispositivos seriais e outros dispositivos com protocolos simples de comunicação. Você voltará a trabalhar com comunicações seriais futuramente, neste livro.

Assuntos e conceitos abordados no capítulo 3:

- arrays e como utilizá-los;
- o que é um potenciômetro (ou resistor variável) e como utilizá-lo;
- como ler valores de voltagem de um pino de entrada analógica;
- como utilizar a função matemática de seno (`sin`);
- como converter graus em radianos;
- o conceito da conversão de uma variável para um tipo diferente;

- Pulse Width Modulation (PWM) e como utilizá-la com `analogWrite()`;
- criação de luzes coloridas utilizando valores RGB diferentes;
- como gerar números aleatórios utilizando `random()` e `randomSeed()`;
- como efeitos distintos de iluminação podem ser gerados com o mesmo circuito, mas com códigos diferentes;
- o conceito de comunicações seriais;
- como definir a taxa de transmissão serial utilizando `Serial.begin()`;
- como enviar comandos utilizando o Serial Monitor;
- o uso de arrays para criar strings de texto;
- como liberar o buffer serial utilizando `Serial.flush`;
- como verificar se dados então sendo enviados pela linha serial utilizando `Serial.available`;
- como criar um loop enquanto uma condição é válida, utilizando o comando `while()`;
- como ler dados da linha serial utilizando `Serial.read()`;
- o conceito básico de ponteiros;
- como enviar dados ao Serial Monitor utilizando `Serial.print()` ou `Serial.printIn()`;
- como manipular strings de texto utilizando a função `strtok()`;
- como converter uma string para um inteiro longo utilizando `strtol()`;
- como restringir o valor de uma variável utilizando a função `constrain()`.

CAPÍTULO 4

Sonorizadores e sensores simples

Este capítulo será bem barulhento. Você conectará um sonorizador piezo ao seu Arduino para adicionar alarmes, avisos sonoros, notificações de alerta etc. ao dispositivo que está criando. Desde a versão 0018 do IDE do Arduino, tons sonoros podem ser adicionados facilmente, graças a um novo comando. Você também verá como utilizar o piezo como um sensor, e aprenderá como ler voltagens a partir dele. Por fim, você aprenderá sobre sensores de luz.

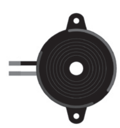
Vamos iniciar com um simples alarme de carro e o comando `tone()`, para emitir sons de seu Arduino.

Projeto 11 – Alarme com sonorizador piezo

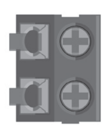
Conectando um sonorizador piezo a um pino de saída digital, você pode criar um som de sirene. Utilizaremos o mesmo princípio do projeto 7, no qual criamos uma lâmpada pulsante empregando uma onda senoidal, mas dessa vez você substituirá o LED por um sonorizador piezo ou disco piezo.

Componentes necessários

Sonorizador piezo (ou disco piezo)



Terminal de parafusos de duas vias



Conectando os componentes

Primeiramente, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue o sonorizador piezo e parafuse seus fios no terminal de parafusos. Conecte o terminal de parafusos à protoboard e depois ao Arduino, como na figura 4.1. Agora, conecte seu Arduino novamente ao cabo USB e, depois, ligue-o.

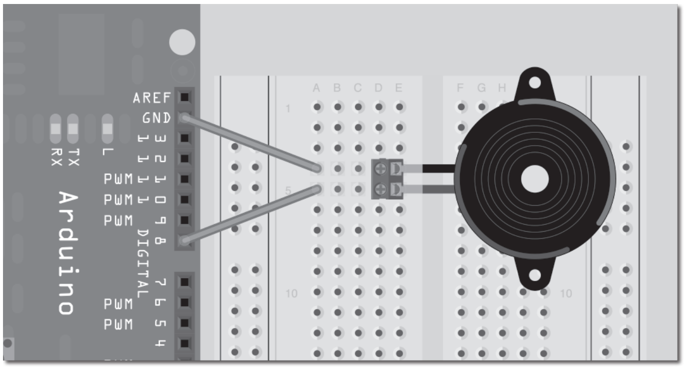


Figura 4.1 – Circuito para o Projeto 11 – Alarme com sonorizador piezo (consulte o site da Novatec para versão colorida).

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 4.1.

Listagem 4.1 – Código para o projeto 11

```
// Projeto 11 - Alarme com sonorizador piezo

float sinVal;
int toneVal;

void setup() {
  pinMode(8, OUTPUT);
}

void loop() {
  for (int x=0; x<180; x++) {
    // converte graus para radianos, e depois obtém o valor do seno
    sinVal = (sin(x*(3.1416/180)));
    // gera uma frequência a partir do valor do seno
    toneVal = 2000+(int(sinVal*1000));
    tone(8, toneVal);
    delay(2);
  }
}
```

Depois do upload do código, haverá uma pequena espera e seu piezo começará a emitir sons. Se tudo estiver funcionando como planejado, você ouvirá um alarme do tipo sirene, como um alarme de carro. O código para o projeto 11 é praticamente idêntico ao do projeto 7; vejamos como ele funciona.

Projeto 11 – Alarme com sonorizador piezo – Análise do código

Primeiramente, você prepara duas variáveis:

```
float sinVal;
int toneVal;
```

A variável de tipo `float`, `sinVal`, armazena o valor de seno que faz o tom se elevar e diminuir, da mesma forma que pulsava a lâmpada do projeto 7. A variável `toneVal` pega o valor de `sinVal` e converte-o para a frequência solicitada.

Na função de inicialização, você define o pino digital 8 como saída:

```
void setup() {
  pinMode(8, OUTPUT);
}
```

No loop principal, você define um loop `for` que irá de 0 a 179 (assim como no projeto 7), para garantir que o valor do seno não fique negativo:

```
for (int x=0; x<180; x++) {
```

Você converte o valor de `x` em radianos (uma vez mais, assim como no projeto 7):

```
sinVal = (sin(x*(3.1412/180)));
```

Então, esse valor é convertido em uma frequência adequada para o som de alarme:

```
toneVal = 2000+(int(sinVal*1000));
```

Você pega `2000`, e adiciona `sinVal` multiplicado por `1000`. Isso fornece um bom intervalo de frequências para o tom crescente e decrescente da onda senoidal.

Depois, você utiliza o comando `tone()` para gerar a frequência no sonorizador piezo:

```
tone(8, toneVal);
```

O comando `tone()` requer dois ou três parâmetros, da seguinte maneira:

```
tone(pin, frequency)
tone(pin, frequency, duration)
```

`pin` corresponde ao pino digital utilizado para emitir a saída ao piezo e `frequency` é a frequência do tom em hertz. Também há o parâmetro opcional `duration`, em milissegundos, para a duração do tom. Se nenhuma duração for especificada, o tom continuará tocando até que você reproduza um tom diferente, ou que utilize o comando `noTone(pin)` para cessar a geração do tom no pino especificado.

Por fim, você espera dois milissegundos entre as alterações de frequência, para garantir que a onda senoidal suba e desça na velocidade desejada:

```
delay(2);
```

Você pode estar se perguntando por que não colocou os dois milissegundos no parâmetro de duração do comando `tone()`, desta forma:

```
tone(8, toneVal, 2);
```

Não o fizemos porque o loop `for` é tão curto que trocará a frequência em menos de dois milissegundos de qualquer forma, tornando inútil o parâmetro de duração. Portanto, uma espera de dois milissegundos é colocada depois que o tom foi gerado, para garantir que ele seja reproduzido por ao menos dois milissegundos, antes que o loop `for` se repita e que o tom se altere novamente.

Você poderá utilizar esse princípio de geração de alarme posteriormente, quando aprender a conectar sensores ao seu Arduino. Então, poderá ativar um alarme quando um limiar de sensor for atingido, como quando alguém se aproxima demais de um detector ultrassônico, ou quando uma temperatura sobe demais.

Alterando os valores de 2.000 e 1.000 no cálculo de `toneVal`, além de alterar o tempo de espera, você pode gerar sons diferentes para o alarme. Divirta-se, e descubra os sons que você pode produzir.

Projeto 11 – Alarme com sonorizador piezo – Análise do hardware

Há dois novos componentes neste projeto: um terminal de parafusos e e um sonorizador piezo. Você utiliza o terminal de parafusos porque os fios de seu sonorizador, ou disco piezo, são finos e moles demais para serem inseridos na protoboard. O terminal de parafusos tem pinos que permitem que ele seja conectado a uma protoboard.

O sonorizador, ou disco piezo, (Figura 4.2) é um dispositivo simples feito de uma fina camada de cerâmica, ligada a um disco metálico.

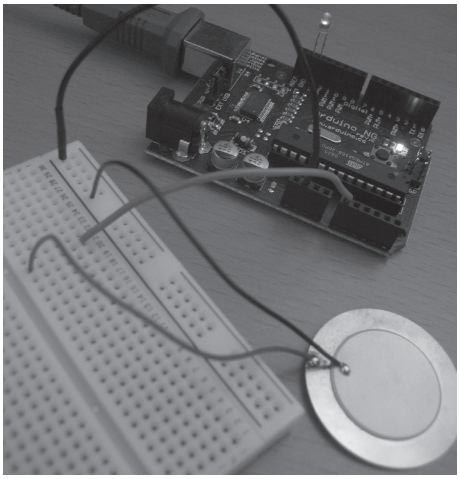


Figura 4.2 – Disco piezo e um Arduino (imagem cortesia de Patrick H. Lauke/splintered.co.uk).

Materiais piezoelétricos, feitos de cristais e cerâmica, têm a capacidade de produzir eletricidade quando uma pressão mecânica é aplicada a eles. Tal efeito tem muitas aplicações úteis, como a produção e detecção de som, a geração de altas voltagens, a geração de frequência eletrônica, o uso em microbalanças e o ajuste ultrafino na precisão de conjuntos ópticos.

O efeito também é reversível: se um campo elétrico for aplicado em um material piezoelétrico, ele fará com que o material mude de forma (em até 0,1%, em alguns casos).

Para produzir sons a partir de um disco piezo, um campo elétrico é ligado e desligado muito rapidamente, fazendo com que o material mude de forma; isso provoca um "clique" audível conforme o disco se deforma (como um pequeno tambor). Alterando a frequência dos pulsos, o disco deformará centenas ou milhares de vezes por segundo, criando um som de zumbido. Alterando a frequência dos cliques e o tempo entre eles, notas específicas podem ser produzidas.

Você também pode utilizar a habilidade do piezo em produzir um campo elétrico para medir movimento ou vibrações. De fato, discos piezo são utilizados como microfones de contato em guitarras e kits de bateria. Você utilizará esse recurso no projeto 13, quando criar um sensor de batida.

Projeto 12 – Tocador de melodia com sonorizador piezo

Em vez de utilizar o piezo para produzir sons irritantes de alarme, por que não usá-lo para tocar uma melodia? Você fará com que seu Arduino toque o refrão da música "Puff, o Dragão Mágico" (no original, "Puff, the Magic Dragon"). Deixe o circuito exatamente como no projeto 11; você alterará apenas o código.

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 4.2.

Listagem 4.2 – Código para o projeto 12

```
// Projeto 12 – Tocador de melodia com sonorizador piezo

#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
```

```
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494

#define WHOLE 1
#define HALF 0.5
#define QUARTER 0.25
#define EIGHTH 0.125
#define SIXTEENTH 0.0625

int tune[] = { NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_B3, NOTE_G3, NOTE_A3,
NOTE_C4, NOTE_C4, NOTE_G3, NOTE_G3, NOTE_F3, NOTE_F3, NOTE_G3, NOTE_F3, NOTE_E3, NOTE_G3,
NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_A3, NOTE_B3, NOTE_C4, NOTE_D4};

float duration[] = { EIGHTH, QUARTER+EIGHTH, SIXTEENTH, QUARTER, QUARTER, HALF, HALF,
HALF, QUARTER, QUARTER, HALF+QUARTER, QUARTER, QUARTER, QUARTER, QUARTER+EIGHTH, EIGHTH,
QUARTER, QUARTER, QUARTER, EIGHTH, EIGHTH, QUARTER, QUARTER, QUARTER, QUARTER,
HALF+QUARTER};

int length;

void setup() {
  pinMode(8, OUTPUT);
  length = sizeof(tune) / sizeof(tune[0]);
}

void loop() {
  for (int x=0; x<length; x++) {
    tone(8, tune[x]);
    delay(1500 * duration[x]);
    noTone(8);
  }
  delay(5000);
}
```

Depois do upload do código, haverá uma pequena espera e então seu piezo começará a tocar uma melodia. Com sorte, você a reconhecerá como parte do refrão de "Puff, o Dragão Mágico". Agora, vejamos os novos conceitos para este projeto.

Projeto 12 – Tocador de melodia com sonorizador piezo – Análise do código

O primeiro elemento que você encontra quando analisa o código para o projeto 12 é a longa lista de diretivas `#define`. A *diretiva #define* é muito simples e útil. `#define` simplesmente define um valor e seu token correspondente. Por exemplo:

```
#define PI 3.14159265358979323846264338327950288419716939937510
```

Isso permitirá que você substitua `PI` em qualquer cálculo, em vez de ter de digitar pi com 50 casas decimais. Outro exemplo:

```
#define TRUE  1
#define FALSE 0
```

Isso significa que você pode colocar `TRUE` ou `FALSE` em seu código, em vez de 0 ou 1, facilitando a leitura humana de instruções lógicas.

Digamos que você tenha escrito certo código para exibir formas em um display de matriz de pontos LED, cuja resolução é 8 x 32. Você poderia criar diretivas `#define` para a altura (*height*) e a largura (*width*) do display da seguinte maneira:

```
#define DISPLAY_HEIGHT 8
#define DISPLAY_WIDTH 32
```

Agora, sempre que você quiser se referir à altura e à largura do display em seu código, poderá utilizar `DISPLAY_HEIGHT` e `DISPLAY_WIDTH`, em vez dos números 8 e 32.

Há duas vantagens principais nesse processo. Primeiro, fica muito mais fácil compreender o código, uma vez que você alterou os valores, transformando-os em tokens que tornam esses números mais claros para terceiros. Em segundo lugar, se você alterar seu display futuramente para uma resolução maior, digamos um display de 16 x 64, bastará alterar os dois valores nas diretivas `#define`, em vez de ter de alterar os números em centenas de linhas de código. Alterando os valores na diretiva `#define` no início do programa, os novos valores serão automaticamente utilizados por todo o restante do código.

No projeto 12, você cria um grande conjunto de diretivas `#define`, em que os tokens são as notas de C3 a B4[1], e os valores são as frequências necessárias para criá-las. A primeira nota de sua melodia é C4, e sua frequência correspondente é 262 Hz. Esse é o C médio na escala musical. (Nem todas as notas definidas são utilizadas em sua melodia, mas foram incluídas, caso você queira compor sua própria música.)

As cinco diretivas `#define` seguintes são utilizadas para definir a duração das notas. As notas podem ter a duração de um compasso inteiro, meio compasso, um quarto, um oitavo ou um dezesseis avos de compasso. Os números serão utilizados para multiplicar

1 N.T.: Os países anglófonos mantiveram a utilização de letras para a nomenclatura das notas musicais. As letras A, B, C, D, E, F e G são utilizadas para as notas lá, si, dó, ré, mi, fá e sol, respectivamente (fonte: Wikipédia).

a duração do compasso, em milissegundos, e obter a duração de cada nota. Por exemplo, como um quarto de nota é 0,25 (ou um quarto de 1), multiplique a duração do compasso (nesse caso, 1.500 milissegundos) por 0,25 para obter a duração de um quarto de nota:

```
1500 x QUARTER = 375 milissegundos
```

Diretivas `#define` também podem ser utilizadas para criar macros; falaremos mais sobre macros em um capítulo futuro.

Em seguida, você define um array inteiro, `tune[]`, preenchendo-o com as notas da melodia "Puff, o Dragão Mágico" da seguinte maneira:

```
int tune[] = { NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_B3, NOTE_G3, NOTE_A3,
NOTE_C4, NOTE_C4, NOTE_G3, NOTE_G3, NOTE_F3, NOTE_F3, NOTE_G3, NOTE_F3, NOTE_E3, NOTE_G3,
NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_A3, NOTE_B3, NOTE_C4, NOTE_D4};
```

Depois disso, você cria outro array de floats, que armazenará a duração de cada nota, conforme ela é tocada:

```
float duration[] = { EIGHTH, QUARTER+EIGHTH, SIXTEENTH, QUARTER, QUARTER, HALF, HALF,
HALF, QUARTER, QUARTER, HALF+QUARTER, QUARTER, QUARTER, QUARTER, QUARTER+EIGHTH, EIGHTH,
QUARTER, QUARTER, QUARTER, EIGHTH, EIGHTH, QUARTER, QUARTER, QUARTER, QUARTER,
HALF+QUARTER};
```

Como você pode ver, analisando esses arrays, o uso das diretivas `#define`, para definir as notas e suas durações, torna a leitura e compreensão do array muito mais fácil do que se tivesse sido preenchido por uma série de números. Em seguida, você cria um inteiro chamado `length`.

```
int length;
```

Esse inteiro será utilizado para calcular e armazenar o comprimento do array (ou seja, o número de notas na melodia).

Em sua rotina de inicialização, você define o pino digital 8 como saída:

```
pinMode(8, OUTPUT);
```

Depois, você inicializa o inteiro `length` com o número de notas no array, utilizando a função `sizeof()`:

```
length = sizeof(tune) / sizeof(tune[0]);
```

A função `sizeof` retorna o número de bytes do elemento passado a ela como parâmetro. No Arduino, um inteiro é feito de dois bytes. Um *byte* é composto de oito bits. (Isso nos conduz ao reino da aritmética binária, mas, para este projeto, você não tem de se preocupar com bits e bytes. Esse tópico será abordado futuramente no livro, e tudo será devidamente explicado.) Sua melodia tem 26 notas, por isso o array `tunes[]` contém 26 elementos. Para esse cálculo, pegamos o tamanho (em bytes) de todo o array:

```
sizeof(tune)
```

E dividimos esse valor pelo número de bytes em um único elemento:

```
sizeof(tune[0])
```

Que, nesse caso, é equivalente a:

```
26 / 2 = 13
```

Se você substituir a melodia no projeto por uma de sua autoria, o comprimento será calculado como o número de notas em sua melodia.

A função `sizeof()` é útil para calcular o comprimento de diferentes tipos de dados, e especialmente adequada se você quiser adaptar seu código para outro dispositivo, em que o comprimento dos tipos de dados seja diferente daqueles do Arduino.

No loop principal você define um loop `for` que itera o número de vezes correspondente às notas na melodia:

```
for (int x=0; x<length; x++) {
```

Então, a próxima nota no array `tune[]` é reproduzida no pino digital 8:

```
tone(8, tune[x]);
```

Depois, você espera o tempo necessário para que a nota seja tocada:

```
delay(1500 * duration[x]);
```

A espera é de 1.500 milissegundos, multiplicada pela duração da nota (0,25 para um quarto de nota, 0,125 para um oitavo etc.).

Antes que a nota seguinte seja tocada, você cessa o tom gerado no pino digital 8.

```
noTone(8);
```

Isso é feito para garantir que, quando duas notas idênticas forem tocadas em sequência, elas possam ser distinguidas como notas individuais. Sem a função `noTone()`, as notas se fundiriam em uma única e longa nota.

Por fim, depois que o loop `for` está completo, você espera cinco segundos antes de repetir a melodia novamente.

```
delay(5000);
```

Para criar as notas dessa melodia, utilizei partituras públicas para "Puff, o Dragão Mágico" que encontrei na Internet, e digitei as notas no array `tune[]`, seguidas pelas durações no array `duration[]`. Repare que adicionei durações diferentes para obter notas pontuadas (por exemplo, `QUARTER+EIGHTH`). Fazendo algo semelhante, você pode criar a melodia que quiser.

Se você quiser acelerar ou desacelerar o ritmo da melodia, altere o valor de 1.500 na função `delay` para algo menor ou maior.

Você também pode substituir o piezo no circuito por um alto-falante ou por fones de ouvido, desde que coloque um resistor em série, para garantir que a corrente máxima do alto-falante não seja excedida.

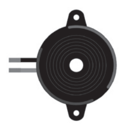
A seguir, você utilizará o disco piezo para outro propósito — sua capacidade em produzir uma corrente quando é apertado ou pressionado. Utilizando esse recurso, você criará um sensor de batida no projeto 13.

Projeto 13 – Sensor de batida piezo

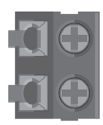
Um disco piezo funciona quando uma corrente elétrica passa pelo material cerâmico do disco, fazendo com que ele mude de forma e produza um som (um clique). O disco também funciona de forma inversa: quando se bate nele ou ele sofre algum tipo de pressão, a força no material provoca a geração de uma corrente elétrica. Você pode ler essa corrente utilizando o Arduino, e é justamente isso que será feito agora, com a criação de um sensor de batida.

Componentes necessários

Sonorizador piezo (ou disco piezo)



Terminal de parafusos de duas vias



LED de 5 mm (de qualquer cor)



Resistor de 1 MΩ



Conectando os componentes

Primeiramente, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Depois, conecte seus componentes como no circuito da figura 4.3. Note que um disco piezo funciona melhor para este projeto do que um sonorizador piezo.

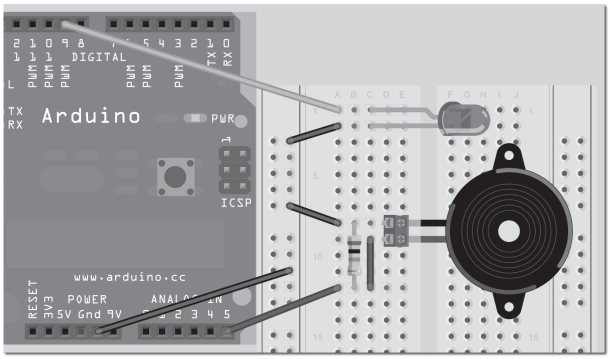


Figura 4.3 – Circuito para o Projeto 13 – Sensor de batida piezo (consulte o site da Novatec para versão colorida).

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 4.3.

Listagem 4.3 – Código para o projeto 13

```
// Projeto 13 - Sensor de batida piezo

int ledPin = 9;        // LED no pino digital 9
int piezoPin = 5;      // Piezo no pino analógico 5
int threshold = 120;   // O valor do sensor a ser atingido antes da ativação
int sensorValue = 0;   // Uma variável para armazenar o valor lido do sensor
float ledValue = 0;    // O brilho do LED


void setup() {
  pinMode(ledPin, OUTPUT);   // Define o ledPin como OUTPUT
  // Pisca o LED duas vezes, para mostrar que o programa iniciou
  digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW); delay(150);
  digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW); delay(150);
}

void loop() {
  sensorValue = analogRead(piezoPin);   // Lê o valor do sensor
  if (sensorValue >= threshold) {     // Se uma batida for detectada, defina o brilho como máximo
    ledValue = 255;
  }
  analogWrite(ledPin, int(ledValue) );     // Escreve o valor de brilho no LED
  ledValue = ledValue - 0.05;     // Apaga o LED lentamenteif (ledValue <= 0) { ledValue = 0;}
// Certifica-se de que o valor não descerá abaixo de zero
}
```

Depois do upload do código, o LED piscará rapidamente duas vezes para indicar que o programa iniciou. Você, agora, pode bater no sensor (primeiro coloque-o sobre uma superfície plana) ou apertá-lo com seus dedos. Toda vez que o Arduino detectar uma batida ou aperto, o LED acenderá, e depois apagará lentamente. (Note que o valor de limiar (`threshold`), no código, foi definido para este disco piezo específico que utilizei na construção do projeto. Talvez seja necessário ajustar esse valor, dependendo do tipo e do tamanho do piezo que você utilizar para seu projeto. Um valor menor é mais sensível, e um maior, menos sensível.)

Projeto 13 – Sensor de batida piezo – Análise do código

Não há comandos novos no código deste projeto, mas analisaremos como ele funciona da mesma forma.

Primeiramente, você prepara as variáveis necessárias para seu programa, que são bem autoexplicativas:

```
int ledPin = 9;         // LED no pino digital 9
int piezoPin = 5;       // Piezo no pino analógico 5
int threshold = 120;    // O valor do sensor a ser atingido antes da ativação
int sensorValue = 0;    // Uma variável para armazenar o valor lido do sensor
float ledValue = 0;     // O brilho do LED
```

Na função de inicialização, o `ledPin` é definido como saída e, como já vimos, faz com que o LED pisque rapidamente duas vezes, para indicar que o programa iniciou sua execução:

```
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW); delay(150);
  digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW); delay(150);
}
```

No loop principal, você primeiro lê o valor analógico do pino analógico 5, ao qual o piezo está conectado:

```
sensorValue = analogRead(piezoPin);
```

Depois, o código verifica se esse valor é maior ou igual (`>=`) ao limiar que você definiu, ou seja, se ele representa uma batida ou pressão. (O piezo é muito sensível, como você perceberá se definir o limiar com um valor muito baixo.) Se afirmativo, o código define `ledValue` como `255`, voltagem máxima do pino digital PWM 9:

```
if (sensorValue >= threshold) {
  ledValue = 255;
}
```

Então, você escreve esse valor no pino digital PWM 9. Como `ledValue` é uma variável `float`, você a converte para um inteiro, uma vez que a função `analogWrite` aceita apenas esse tipo de dado:

```
analogWrite(ledPin, int(ledValue));
```

Na sequência, você reduz o valor de `ledValue`, que é um `float`, em `0.05`:

```
ledValue = ledValue - 0.05;
```

Você deseja que o LED apague lentamente, por isso utiliza um valor de ponto flutuante, em vez de um inteiro, para armazenar o valor do brilho do LED. Dessa forma, você pode reduzir seu valor em pequenos decrementos (nesse caso, 0,05), de modo que demore um pouco, enquanto o loop principal se repete, para que o valor de `ledValue` atinja zero. Se você quiser que o LED apague mais lentamente ou mais rapidamente, altere esse valor.

Por fim, você não deseja que `ledValue` seja menor que zero, uma vez que o pino digital PWM 9 pode emitir apenas um valor de 0 a 255, por isso deve verificar se o valor é menor ou igual a zero e, caso seja, alterá-lo novamente para zero:

```
if (ledValue <= 0) { ledValue = 0;}
```

Depois disso, o loop principal se repete, apagando o LED lentamente a cada execução, até que ele apague completamente, ou que seja detectada outra batida, definindo novamente o brilho como máximo.

Agora, vamos apresentar um novo sensor, o resistor dependente de luz (Light Dependent Resistor, ou LDR).

Projeto 14 – Sensor de luz

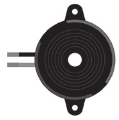
Este projeto apresenta um novo componente, conhecido como resistor dependente de luz, ou LDR. Como implica seu nome, esse dispositivo é um resistor que depende de luz. Em um ambiente escuro, o resistor tem uma resistência muito alta. Conforme fótons (luz) atingem o detector, a resistência diminui. Quanto mais luz, menor a resistência. Lendo o valor do sensor, você pode detectar se o ambiente está claro, escuro, ou com qualquer variação de luminosidade. Neste projeto, você utilizará um LDR para detectar a luz, e um sonorizador piezo para fornecer feedback sonoro da quantidade de luz detectada.

Essa configuração poderia, por exemplo, ser utilizada em um alarme que indica quando uma porta foi aberta. Da mesma forma, você também poderia utilizá-la para criar um instrumento musical semelhante a um teremim[2].

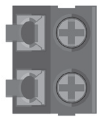
2 N.T.: O teremim é um dos primeiros instrumentos musicais completamente eletrônicos. Ele é único, por não precisar de nenhum contato físico para produzir música, pois é executado movimentando-se as mãos no ar (fonte: Wikipédia).

Componentes necessários

Sonorizador piezo (ou disco piezo)



Terminal de parafusos de duas vias



Resistor dependente de luz



Resistor de 10 kΩ



Conectando os componentes

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Depois, conecte seus componentes como no circuito da figura 4.4. Verifique todas as suas conexões antes de reconectar a energia ao Arduino.

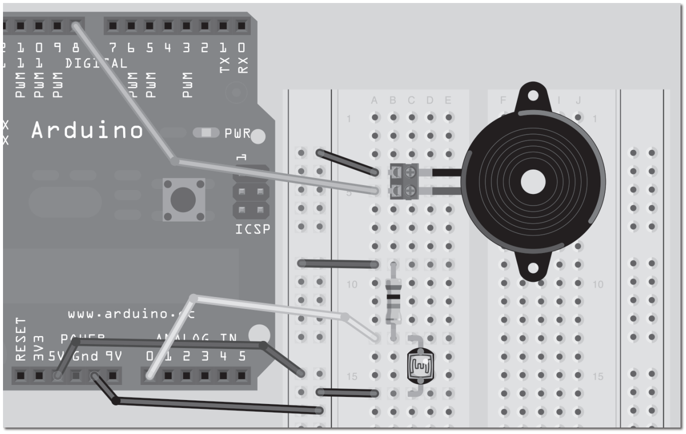


Figura 4.4 – Circuito para o Projeto 14 – Sensor de luz (consulte o site da Novatec para versão colorida).

O LDR pode ser inserido de qualquer forma, uma vez que não tem polaridade. Em meu caso, percebi que um resistor de 10 kΩ teve desempenho satisfatório, mas você pode ter de experimentar configurações diferentes de resistor até que encontre uma adequada ao seu LDR. Um valor entre 1 kΩ e 10 kΩ deve ser o suficiente. Ter opções de resistores de valores diferentes em sua caixa de componentes é sempre uma boa opção.

Digite o código

Ligue o IDE de seu Arduino, e digite o código simples e curto da listagem 4.4.

Listagem 4.4 – Código para o projeto 14

```
// Projeto 14 - Sensor de luz

int piezoPin = 8; // Piezo no pino 8
int ldrPin = 0;   // LDR no pino analógico 0
int ldrValue = 0; // Valor lido do LDR

void setup() {
  // nada a ser feito aqui
}

void loop() {
    ldrValue = analogRead(ldrPin);   // lê o valor do LDR
    tone(piezoPin,1000);             // toca um tom de 1000 Hz do piezo
    delay(25);                       // espera um pouco
    noTone(piezoPin);                // interrompe o tom
    delay(ldrValue);                 // espera a quantidade de milissegundos em ldrValue
}
```

Quando você fizer o upload desse código, o Arduino reproduzirá pequenos bips. O intervalo entre os bips será longo se o LDR estiver na sombra, e curto se houver luz brilhando sobre ele, fornecendo um efeito semelhante ao de um contador Geiger. Pode ser mais prático soldar um conjunto de fios longos ao LDR, para permitir que você mantenha sua protoboard e seu Arduino na mesa enquanto movimenta seu LDR, apontando-o para regiões escuras e claras. Como alternativa, ilumine o sensor com uma lanterna e movimente a luz sobre ele.

O código para o projeto 14 é muito simples, e você deve ser capaz de descobrir sozinho como ele funciona, sem minha ajuda. Entretanto, mostrarei como um LDR funciona e por que o resistor adicional é importante.

Projeto 14 – Sensor de luz – Análise do hardware

O novo componente neste projeto é um resistor dependente de luz (Light Dependent Resistor, ou LDR), também conhecido como um CdS (*Cadmium-Sulfide*, ou sulfeto de cádmio), ou fotorresistor. LDRs têm diferentes formas e tamanhos (Figura 4.5), e diferentes valores de resistência.

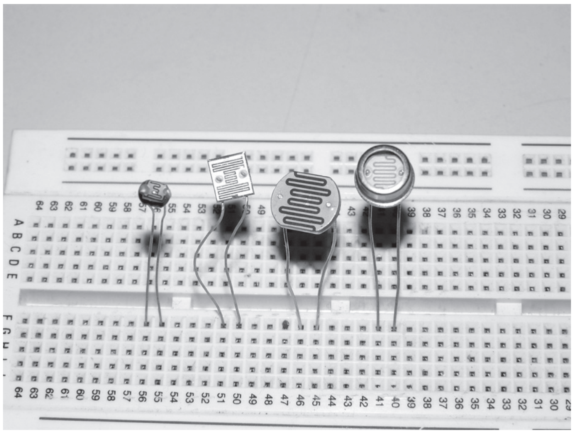


Figura 4.5 – Tipos diferentes de LDRs (imagem por cultured_society2nd).

Cada um dos terminais do LDR vai para um eletrodo. Entre um material mais escuro, formando uma linha sinuosa entre os eletrodos, está o material fotocondutor. O componente tem um revestimento de plástico transparente ou vidro. Quando a luz atinge o material fotocondutor, ele perde sua resistência, permitindo que mais corrente flua entre os eletrodos. LDRs podem ser utilizados em todos os tipos de projetos interessantes; por exemplo, você poderia disparar um laser em um LDR e detectar quando uma pessoa interrompe o feixe, disparando um alarme ou o obturador de uma câmera.

O próximo conceito em seu circuito é um divisor de tensão (também conhecido como divisor de potencial). É aqui que o resistor entra em cena. Utilizando dois resistores e passando a voltagem por apenas um deles, você pode reduzir a voltagem que entra no circuito. Em seu caso, você tem um resistor de valor fixo (10 kΩ ou algo do tipo) e um resistor variável na forma de um LDR. Vamos analisar um divisor de tensão padrão, utilizando resistores para ver como isso funciona. A figura 4.6 mostra um divisor de voltagem utilizando dois resistores.

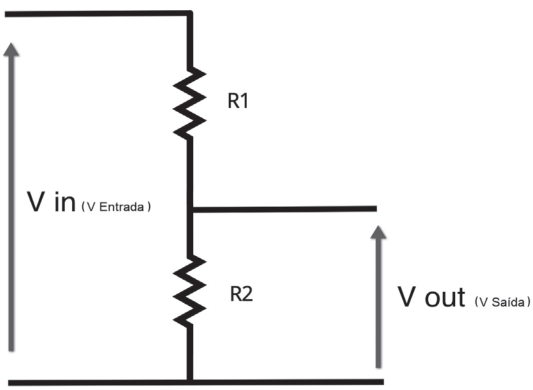


Figura 4.6 – Divisor de tensão.

A voltagem de entrada (V_{in}) está conectada a ambos os resistores. Ao medir a voltagem em um dos resistores (V_{out}), ela será menor (dividida). A fórmula para calcular qual será a voltagem em V_{out}, quando medida em R2, é:

$$V_{out} = \frac{R2}{R2 + R1} \times V_{in}$$

Assim, se você tem resistores de 100 Ω (ou 0,1 kΩ) para R1 e R2, e 5 V entrando em V_{in}, sua fórmula é:

$$\frac{0{,}1}{0{,}1 + 0{,}1} \times 5 = 2{,}5\ V$$

Repita o processo com resistores de 470 Ω:

$$\frac{0{,}47}{0{,}47 + 0{,}47} \times 5 = 2{,}5\ V$$

Uma vez mais, você tem 2,5 V. Isso demonstra que o valor dos resistores não é importante, mas, sim, a razão entre eles. Vamos tentar com um resistor de 1 kΩ e outro de 500 Ω:

$$\frac{0{,}5}{0{,}5 + 1} \times 5 = 1{,}66\ V$$

Com o resistor de baixo tendo metade do valor do resistor de cima, você tem 1,66 V, que é um terço da voltagem de entrada. Vamos fazer com que o resistor de baixo tenha o dobro do valor do resistor de cima, ou seja, 2 kΩ:

$$\frac{2}{2 + 1} \times 5 = 3{,}33\ V$$

Isso representa dois terços da voltagem de entrada. Assim, vamos aplicar tudo isso ao LDR. Você pode presumir que o LDR tenha um alcance de cerca de 100 kΩ no escuro e 10 kΩ quando colocado na claridade. A tabela 4.1 mostra as voltagens que você verá no circuito, conforme a resistência se altera.

Tabela 4.1 – Valores de Vout (voltagem de saída) para um LDR com 5 V como Vin (voltagem de entrada)

R1	R2 (LDR)	Vout	Claridade
10 kΩ	100 kΩ	4,54 V	Mais escuro
10 kΩ	73 kΩ	4,39 V	25%
10 kΩ	45 kΩ	4,09 V	50%
10 kΩ	28 kΩ	3,68 V	75%
10 kΩ	10 kΩ	2,5 V	Mais claro

Como você pode ver, à medida que a claridade aumenta, a voltagem em Vout diminui. Como resultado, o valor que você lê no sensor também diminui, e o intervalo entre dois bips se torna mais curto, fazendo com que os bips ocorram mais frequentemente. Se você trocasse a posição do resistor com a do LDR, a voltagem aumentaria, à medida que mais luz atingisse o LDR. As duas formas funcionam, tudo depende da leitura que você deseja em seu sensor.

Resumo

No capítulo 4, você aprendeu como criar música, sons de alarme, avisos sonoros etc. com seu Arduino. Esses recursos têm muitas aplicações úteis. Você pode, por exemplo, criar seu próprio despertador. Utilizando um sonorizador piezo de modo inverso, para detectar voltagens, e empregando esse efeito para detectar uma batida ou pressão no disco, você pode criar um instrumento musical. Por fim, utilizando um LDR para detectar luminosidade, você pode acender uma lâmpada quando a luz ambiente estiver abaixo de determinado limiar.

Assuntos e conceitos abordados no capítulo 4:

- o que é um transdutor piezoelétrico e como ele funciona;
- como criar sons utilizando a função `tone()`;
- como interromper a geração de um tom utilizando a função `noTone()`;
- a diretiva `#define` e como ela facilita a depuração e o entendimento do código;
- como obter o tamanho de um array (em bytes) utilizando a função `sizeof()`;
- o que é LDR (Light Dependent Resistor), como ele funciona e como ler seus valores;
- o conceito de divisores de tensão e como utilizá-los.

CAPÍTULO 5

Controlando um motor CC

Chegou o momento de controlarmos um motor CC (*DC motor*, ou motor de corrente contínua). Se você planeja construir um robô ou qualquer tipo de dispositivo móvel, as habilidades que está prestes a aprender serão essenciais. Controlar um motor requer correntes mais altas do que as que o Arduino pode produzir com segurança em suas saídas, por isso você terá de utilizar transistores para garantir corrente suficiente para o motor, e diodos para proteção do Arduino. A análise do hardware explicará como esses elementos funcionam.

Para seu primeiro projeto, você controlará um motor utilizando um método muito simples. Depois, utilizará o popular chip controlador de motor L293D. Futuramente, você aprenderá como utilizar esses elementos para controlar um motor de passo.

Projeto 15 – Controle de um motor simples

Primeiramente, você apenas controlará a velocidade de um motor CC em uma direção, utilizando um transistor de potência, um diodo, uma fonte de alimentação externa (para fornecer energia ao motor) e um potenciômetro (para controlar a velocidade). Qualquer transistor de potência NPN adequado, projetado para correntes elevadas, pode substituir o transistor TIP120.

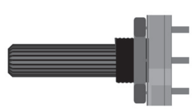
A fonte de alimentação externa pode ser um par de pilhas ou uma fonte externa para alimentação CC; ela deve ter voltagem e corrente suficientes para controlar o motor. A voltagem não deve exceder aquela exigida pelo motor. Para nossos testes, utilizei uma fonte de alimentação CC que forneceu 5 V a 500 mA, suficiente para o motor CC de 5 V que utilizamos. Note que, se você utilizar uma fonte de alimentação com voltagem maior do que a que o motor é capaz de receber, poderá danificá-lo permanentemente.

Componentes necessários

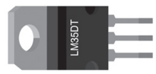
Motor CC



Potenciômetro de 10 kΩ



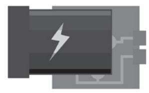
Transistor TIP120 *



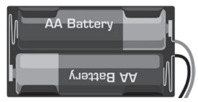
Diodo 1N4001 *



Plugue fêmea



Fonte de alimentação externa



* Ou um equivalente adequado.

Conectando os componentes

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes necessários e conecte-os como na Figura 5.1. É essencial que você verifique com muita calma se todas as suas conexões estão corretas, antes de fornecer energia ao circuito. Qualquer erro pode resultar em danos aos seus componentes ou ao Arduino. O diodo desempenha um papel essencial na proteção do Arduino contra a força eletromotriz inversa, a qual explicarei futuramente.

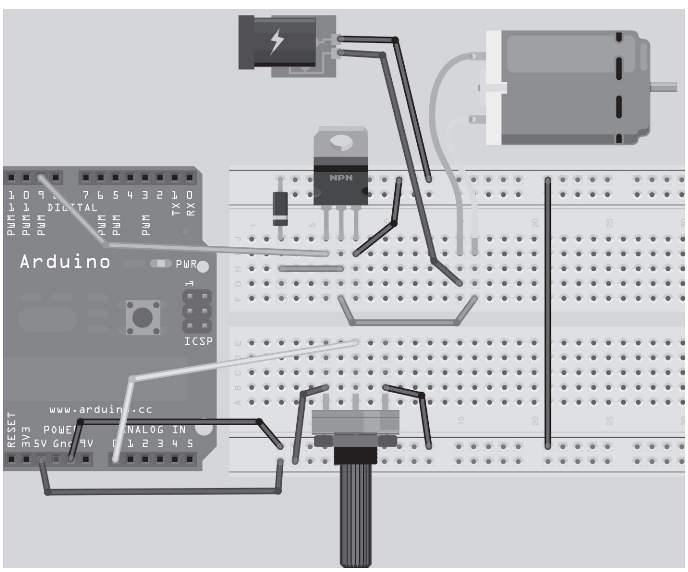


Figura 5.1 – Circuito para o Projeto 15 – Controle de um motor simples (consulte o site da Novatec para versão colorida).

Digite o código

Abra o IDE de seu Arduino e digite o código da listagem 5.1. Antes de fazer seu upload, desconecte a fonte de alimentação externa do motor e verifique se o potenciômetro está girado até o fim, em sentido horário. Agora, faça o upload do código para o Arduino.

Listagem 5.1 – Código para o projeto 15

```
// Projeto 15 - Controle de um motor simples

int potPin = 0;          // Analógico no 0, conectado ao potenciômetro
int transistorPin = 9;   // Pino PWM 9 conectado à base do transistor
int potValue = 0;        // valor retornado do potenciômetro

void setup() {
  // define o pino do transistor como saída:
  pinMode(transistorPin, OUTPUT);
}

void loop() {
  // lê o potenciômetro, converte para o intervalo de 0 a 255:
  potValue = analogRead(potPin) / 4;
  // utilize isso para controlar o transistor:
  analogWrite(transistorPin, potValue);
}
```

Depois de fazer o upload do código, conecte a fonte de alimentação externa. Agora você pode virar o potenciômetro para controlar a velocidade do motor.

Projeto 15 – Controle de um motor simples – Análise do código

Primeiramente, declare as três variáveis que armazenarão: o valor para o pino analógico conectado ao potenciômetro, o pino PWM conectado à base do transistor, e uma terceira variável para armazenar o valor lido do potenciômetro, a partir do pino analógico 0:

```
int potPin = 0;          // Analógico no 0, conectado ao potenciômetro
int transistorPin = 9;   // Pino PWM 9 conectado à base do transistor
int potValue = 0;        // valor retornado do potenciômetro
```

Na função `setup()`, você define o `pinMode` do pino do transistor como saída:

```
void setup() {
  // define o pino do transistor como saída:
  pinMode(transistorPin, OUTPUT);
}
```

No loop principal, `potValue` é definida com o valor lido no pino analógico 0 (o `potPin`) e, depois, é dividida por `4`:

```
potValue = analogRead(potPin) / 4;
```

Essa operação é necessária, pois o valor analógico irá de 0 (para 0 V) a 1023 (para 5 V). O valor que você deve escrever no pino do transistor pode ir apenas de 0 a 255, por isso você divide o valor do pino analógico 0 (1023, no máximo) por 4, obtendo um valor máximo de 255, atribuível ao pino digital 9 (utilizando `analogWrite`, uma vez que você está usando PWM).

Em seguida, o código escreve no pino do transistor o valor do potenciômetro:

```
analogWrite(transistorPin, potValue);
```

Em outras palavras, quando você gira o potenciômetro, valores diferentes, indo de 0 a 1023, são lidos; esses valores são convertidos para o intervalo entre 0 e 255. Então, o valor é escrito (via PWM) no pino digital 11, alterando a velocidade do motor CC. Gire o potenciômetro totalmente para a direita e o motor desliga, gire para a esquerda e o motor acelera até que atinja a velocidade máxima. O código é muito simples e não deve lhe apresentar nenhuma novidade.

Vamos, agora, examinar o hardware utilizado neste projeto, e descobrir como tudo funciona.

Projeto 15 – Controle de um motor simples – Análise do hardware

O circuito é essencialmente dividido em duas seções. A seção um é o potenciômetro, conectado aos 5 V e ao terra, com o pino central indo para o pino analógico 0. Conforme você gira o potenciômetro a resistência se altera, para permitir que voltagens de 0 V a 5 V saiam do pino central, cujo valor é lido utilizando o pino analógico 0.

A segunda seção é a que controla a alimentação do motor. Os pinos digitais no Arduino fornecem no máximo 40 mA (miliampères). Um motor CC pode exigir cerca de 500 mA para operar em velocidade máxima; esse é obviamente um valor alto demais para o Arduino. Se você tentasse controlar o motor diretamente a partir de um pino do Arduino, danos sérios e permanentes poderiam acontecer.

Portanto, você tem de encontrar uma forma de fornecer uma corrente mais alta a ele. A solução é utilizar uma fonte de alimentação externa, a qual fornecerá corrente suficiente para alimentar o motor. Você poderia utilizar a saída de 5 V do Arduino, capaz de fornecer até 800 mA, quando conectada a uma fonte de alimentação externa. Entretanto, placas do Arduino são caras, e é muito fácil danificá-las quando conectadas a fontes de correntes elevadas, como motores CC. Por segurança, é melhor utilizar uma fonte de alimentação externa. Da mesma forma, seu motor pode exigir 9 V ou 12 V, ou amperagens mais altas, e isso está além do que o Arduino pode fornecer.

Note também que este projeto controla a velocidade do motor, por isso você necessita de uma forma de controlar essa voltagem para acelerar ou desacelerar o motor. É para isso que utilizaremos o transistor TIP-120.

Transistores

Um transistor é essencialmente um interruptor digital, que também pode ser utilizado como um amplificador de potência. Em seu circuito, você o utilizará como uma chave. O símbolo eletrônico para um transistor pode ser visto na figura 5.2.

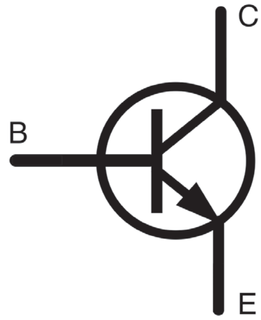


Figura 5.2 – Símbolo para um transistor NPN.

O transistor tem três terminais: a Base, o Coletor e o Emissor, marcados como B, C e E no diagrama. Em seu circuito, você tem até 5 V entrando na Base por meio do pino digital 9. O Coletor está conectado a um terminal no motor. O Emissor está conectado ao terra. Sempre que você aplica uma voltagem à Base por meio do pino digital 9, o transistor liga, permitindo que a corrente flua por ele, entre o Emissor e o Coletor e, assim, alimentando o motor que está conectado em série com esse circuito. Aplicando uma pequena corrente à Base, você pode controlar uma corrente mais elevada entre o Emissor e o Coletor.

Transistores são componentes essenciais em praticamente qualquer equipamento eletrônico moderno. Muitas pessoas consideram transistores como a maior invenção do século 20. Processadores de PCs e laptops têm entre 300 milhões e 1 bilhão de transistores.

Em nosso caso, você utilizou o transistor como uma chave para ligar e desligar uma voltagem e uma corrente mais altas. Quando uma corrente é aplicada à Base, a voltagem aplicada ao Coletor é ligada e permite-se que ela flua entre o Coletor e o Emissor. Como você está pulsando seu sinal, o transistor liga e desliga muitas vezes por segundo; é essa corrente pulsada que controla a velocidade do motor.

Motores

Um motor é um eletromagneto, que possui um campo magnético enquanto energia é fornecida a ele. Quando a alimentação é removida, o campo magnético colapsa; esse processo pode produzir uma tensão reversa, que retorna em sua fiação. Isso poderia danificar seriamente seu Arduino, e é por isso que o diodo foi posicionado de forma contrária no circuito. A listra branca no diodo normalmente vai para o terra. Energia fluirá do lado positivo para o lado negativo. Como você posicionou o diodo ao contrário, nenhuma energia o atravessará. Entretanto, se o motor produzir uma força eletromotriz inversa e enviar corrente de volta pela fiação, o diodo atuará como uma válvula para impedir que isso aconteça. Dessa forma, o diodo em seu circuito foi adicionado para proteger seu Arduino.

Se você conectasse um motor CC diretamente a um multímetro sem nenhum outro componente conectado, e girasse o eixo do motor, veria que o motor gera uma corrente. É exatamente dessa forma que turbinas funcionam. Quando você remove a alimentação de um motor que está funcionando e rotacionando, o motor continua girando em razão de sua inércia até que pare. Nesse intervalo de tempo, em que o motor rotaciona sem alimentação aplicada, ele gera uma corrente. Essa é a mencionada força eletromotriz inversa (*back EMF*). Novamente, o diodo atua como uma válvula e garante que a eletricidade não flua contrariamente pelo circuito, danificando outros componentes.

Diodos

Diodos são válvulas de uma via. Exatamente da mesma forma que uma válvula antirretorno em um encanamento permite que a água flua em apenas uma direção, o diodo permite que uma corrente flua em uma direção, mas não em outra. Você encontrou diodos quanto utilizou LEDs. Um LED tem uma polaridade. Você conecta o terminal positivo da alimentação ao terminal mais comprido do LED para acendê-lo. Posicionando-o ao contrário, o LED não apenas não acenderá, como também impedirá que a eletricidade flua em seus terminais. O diodo tem uma faixa branca ao seu redor, próxima do terminal negativo. Imagine essa faixa branca como uma barreira. Eletricidade flui pelo diodo a partir do terminal que não tem barreira. Quando você reverte a voltagem e tenta fazer com que ela flua pelo lado que tem a faixa, a corrente é interrompida.

Diodos são essenciais na proteção de circuitos contra tensão reversa, como quando você conecta uma fonte de alimentação da forma errada, ou se uma voltagem é revertida, como uma força eletromotriz inversa em seu circuito. Dessa forma, procure sempre utilizá-los em seus circuitos quando houver risco de que a alimentação seja revertida, seja por erro no uso ou por fenômenos como a força eletromotriz inversa.

Projeto 16 – Uso do CI controlador de motor L293D

No projeto anterior, você utilizou um transistor para controlar o motor. Neste projeto, você utilizará um CI (circuito integrado) controlador de motor muito popular, o L293D. A vantagem do uso desse chip é que você pode controlar dois motores ao mesmo tempo, além de poder controlar sua direção. O chip também pode ser utilizado para controlar um motor de passo, como você descobrirá no projeto 28. (Você também pode utilizar um chip compatível pino a pino, o SN754410, que tem uma taxa de corrente mais alta.) Sentiu falta de algo na lista de componentes? Diodos, talvez? Não se preocupe, o CI tem seus próprios diodos internos, por isso você não necessita de um para este projeto.

Componentes necessários

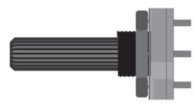
Motor CC



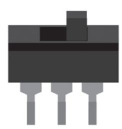
CI controlador de motor L293D ou SN754410



Potenciômetro de 10 kΩ



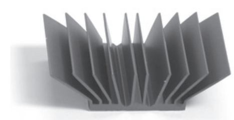
Chave seletora



Resistor de 10 kΩ



Dissipador de calor



Conectando os componentes

Primeiramente, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes necessários e conecte-os como na figura 5.3. Uma vez mais, verifique o circuito calmamente antes de ligá-lo. O L293D fica MUITO quente quando utilizado. Portanto, um dissipador de calor é essencial.

Cole o dissipador no topo do chip utilizando uma cola epóxi forte. Quanto maior o dissipador de calor, melhor. Saiba que a temperatura pode subir a ponto de derreter o plástico de uma protoboard, ou os fios que a toquem. Não toque no dissipador de calor para não se queimar. Não deixe o circuito ligado ou sem supervisão, caso ele superaqueça. Pode ser prudente utilizar uma stripboard, em vez de uma protoboard, neste projeto, para evitar danificar sua protoboard em razão do calor.

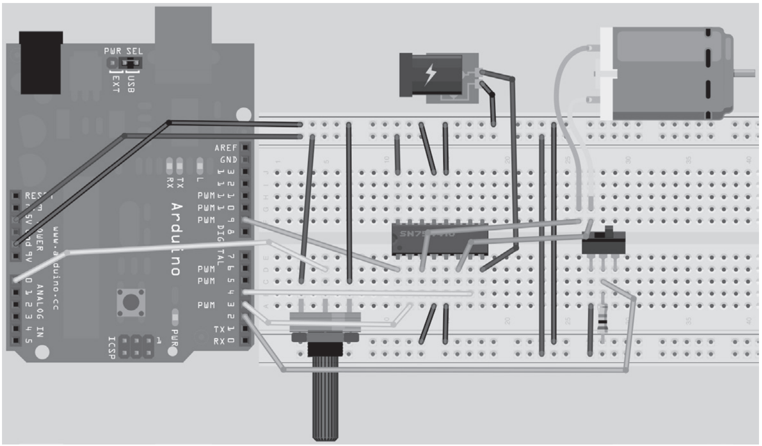


Figura 5.3 – Circuito para o Projeto 16 (consulte o site da Novatec para versão colorida).

Digite o código

Assim que você estiver convencido de que seu circuito foi conectado corretamente, faça o upload do código da listagem 5.2. Ainda não conecte a fonte de alimentação externa.

Listagem 5.2 – Código para o projeto 16

```
// Projeto 16 - Uso do CI controlador de motor L293D
#define switchPin 2   // Entrada da chave
#define motorPin1 3   // Entrada 1 do L293D
#define motorPin2 4   // Entrada 2 do L293D
#define speedPin 9    // Pino de Ativação 1 do L293D
#define potPin 0      // Potenciômetro no pino analógico 0
int Mspeed = 0;       // Uma variável para armazenar o valor de velocidade atual

void setup() {
//define o pino da chave como INPUT
pinMode(switchPin, INPUT);

// define os pinos remanescentes como saída
pinMode(motorPin1, OUTPUT);
pinMode(motorPin2, OUTPUT);
pinMode(speedPin, OUTPUT);
}
```

```
void loop() {
  Mspeed = analogRead(potPin)/4;      // lê o valor de velocidade a partir do potenciômetro
  analogWrite(speedPin, Mspeed);      // escreve a velocidade para o pino de Ativação 1
  if (digitalRead(switchPin)) {       // Se a chave estiver HIGH, gire o motor em sentido horário
    digitalWrite(motorPin1, LOW);     // define a Entrada 1 do L293D como baixa
    digitalWrite(motorPin2, HIGH);    // define a Entrada 2 do L293D como alta
  }
  else {  // se a chave estiver LOW, gire o motor em sentido anti-horário
    digitalWrite(motorPin1, HIGH);    // define a Entrada 1 do L293D como baixa
    digitalWrite(motorPin2, LOW);     // define a Entrada 2 do L293D como alta
  }
}
```

Assim que tiver terminado o upload do código, ajuste o potenciômetro em seu ponto médio e conecte a fonte de alimentação externa. O motor agora deverá girar; você pode ajustar sua velocidade regulando o potenciômetro. Para alterar a direção do motor, primeiro ajuste a velocidade no mínimo, então altere a chave seletora. O motor agora rotacionará na direção oposta. Lembre-se, tenha cuidado com o chip, pois ele fica muito quente quando ligado.

Projeto 16 – Uso do CI controlador de motor L293D – Análise do código

O código para este projeto é muito simples. Primeiramente, você define os pinos que vai utilizar no Arduino:

```
#define switchPin 2   // Entrada da chave
#define motorPin1 3   // Entrada 1 do L293D
#define motorPin2 4   // Entrada 2 do L293D
#define speedPin 9    // Pino de ativação 1 do L293D
#define potPin 0      // Potenciômetro no pino analógico 0
```

Depois, define um inteiro para armazenar o valor de velocidade lido do potenciômetro:

```
int Mspeed = 0;  // Uma variável para armazenar o valor de velocidade atual
```

Na função de inicialização, você define os pinos apropriados como entradas ou saídas:

```
pinMode(switchPin, INPUT);

pinMode(motorPin1, OUTPUT); // Do ponto de vista do Arduino, esses três pinos são usados como saídas
pinMode(motorPin2, OUTPUT);
pinMode(speedPin, OUTPUT);
```

No loop principal, você primeiro lê o valor do potenciômetro conectado ao pino analógico 0, armazenando-o em `Mspeed`:

```
Mspeed = analogRead(potPin)/4; // lê o valor de velocidade a partir do potenciômetro
```

Depois, você define o valor PWM do pino PWM 9 com a velocidade apropriada:

```
analogWrite(speedPin, Mspeed); // escreve a velocidade para o pino de Ativação 1
```

Na sequência, temos uma instrução `if` para decidir se o valor lido do pino da chave é `HIGH` ou `LOW`. Se ele for `HIGH`, então a Entrada 1 do L293D será definida como `LOW`, e a Entrada 2 como `HIGH`. Isso é o mesmo que a Entrada 2 ter uma tensão positiva e a Entrada 1 ser o terra, fazendo com que o motor gire em uma direção:

```
if (digitalRead(switchPin)) {     // Se a chave estiver HIGH, gire o motor em sentido horário
  digitalWrite(motorPin1, LOW);  // define a Entrada 1 do L293D como baixa
  digitalWrite(motorPin2, HIGH); // define a Entrada 2 do L293D como alta
}
```

Se o pino da chave estiver `LOW`, então a Entrada 1 do L293D será definida como `HIGH`, e a Entrada 2 como `LOW`, revertendo a direção do motor:

```
else { // se a chave estiver LOW, gire o motor em sentido anti-horário
  digitalWrite(motorPin1, HIGH);   // define a Entrada 1 do L293D como alta
  digitalWrite(motorPin2, LOW);    // define a Entrada 2 do L293D como baixa
}
```

O loop repetirá, verificando se há um novo valor de velocidade ou uma nova direção, e definindo os pinos apropriados de velocidade e direção. Como você pode ver, utilizar o CI controlador de motor não é tão difícil quanto pode parecer a princípio. De fato, isso facilitou muito sua vida, pois recriar o circuito e o código que vimos sem o chip seria muito mais complexo. Nunca se assuste com CIs! Uma leitura calma e cuidadosa de seus datasheets deve revelar seus segredos. Vejamos como funcionam os componentes apresentados neste projeto.

Projeto 16 – Uso do CI controlador de motor L293D – Análise do hardware

O novo componente no projeto 16 é um CI controlador de motor, que pode ser o L293D ou o SN754410, dependendo de sua escolha (há outros chips disponíveis, e um pouco de pesquisa na Internet revelará outros controladores de pinos compatíveis).

O L293D é o que chamamos de Ponte H Dupla. Uma ponte H é um conceito eletrônico muito útil, ainda que simples (Figura 5.4).

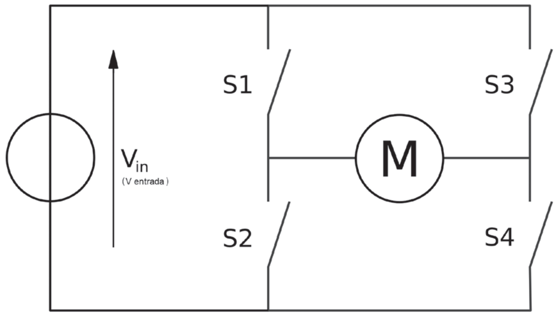


Figura 5.4 – Ponte H com interruptores (imagem por Cyril Buttay).

Na figura 5.4, um motor está conectado a quatro chaves. Essa configuração é conhecida como uma ponte H, pois lembra uma letra H, com a ponte de carga no centro. Agora, analise a figura 5.5.

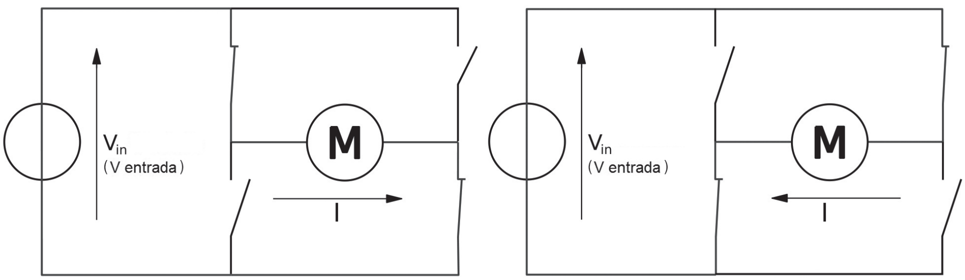


Figura 5.5 – Alteração da direção do motor na ponte H (imagem por Cyril Buttay).

À esquerda, as chaves do canto superior esquerdo e do canto inferior direito estão fechadas. Dessa forma, a corrente fluirá pelo motor, da esquerda para a direita, e o motor rotacionará. Caso você abra essas chaves e feche as chaves do canto superior direito e do canto inferior esquerdo, a corrente fluirá pelo motor na direção oposta, fazendo com que ele rotacione também na direção oposta. É dessa forma que funciona uma ponte H.

O CI controlador do motor é formado de duas pontes H. Em vez de chaves, ele utiliza transistores. Da mesma forma que o transistor do projeto 15 foi utilizado como chave, os transistores dentro da ponte H abrem e fecham na mesma configuração da figura 5.5, para que seu motor rotacione nas duas direções. O chip é uma ponte H dupla, pois tem duas pontes H. Dessa forma, você é capaz de controlar dois motores, e suas direções, ao mesmo tempo.

Note que você poderia criar sua própria ponte H com transistores e diodos, e ela desempenharia a mesma função do L293D; mas lembre-se de que o L293D tem a vantagem de ser pequenino, e de oferecer todos aqueles transistores e diodos em um pequeno espaço.

EXERCÍCIO

Agora, experimente o seguinte exercício:

- **Exercício 1**: Em vez de apenas um motor, tente conectar dois, com duas chaves de direção e potenciômetros para controlar a direção e a velocidade. A figura 5.6 mostra a pinagem do chip.

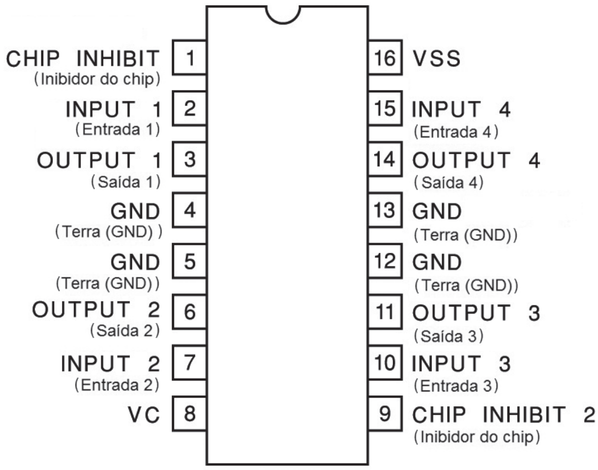


Figura 5.6 – Pinagem para o L293D (ou SN754410).

Resumo

O capítulo 5 mostrou-lhe como utilizar motores em seus projetos. Também apresentou os conceitos importantes de transistores e diodos; você utilizará muito esses componentes em seus projetos, especialmente para controlar dispositivos que exijam voltagens ou correntes mais altas do que as que o Arduino pode fornecer.

Agora você também sabe como construir uma ponte H e como utilizá-la para alterar a direção de um motor. Você também utilizou o popular CI controlador de motor L293D; você trabalhará com esse chip futuramente no livro, quando encontrar outro tipo de motor. As habilidades necessárias para controlar um motor são vitais se você pretende construir um robô, ou um veículo de controle remoto de qualquer tipo, utilizando o Arduino.

Assuntos e conceitos abordados no capítulo 5:

- o uso de um transistor para fornecer alimentação a dispositivos;
- como controlar a velocidade de um motor utilizando PWM;
- a corrente fornecida por um pino digital do Arduino;
- como utilizar uma fonte de alimentação externa, essencial para dispositivos de alta potência;
- o conceito de transistores e como eles funcionam;
- motores e como eles funcionam;
- como utilizar um diodo para evitar a força eletromotriz inversa;
- como funciona um diodo;
- como motores podem ser utilizados para gerar potência;
- como diodos são vitais na proteção de um circuito;
- como utilizar um motor com um controlador de motor L293D;
- por que dissipadores de calor são essenciais para dissipar o calor de CIs;
- o conceito de uma ponte H e como ela pode ser utilizada para alterar a direção de um motor.

CAPÍTULO 6

Contadores binários

Você agora voltará a trabalhar com LEDs. Dessa vez, entretanto, não os controlará diretamente a partir do Arduino. Em vez disso, você utilizará um incrível chip, conhecido como *registrador de deslocamento*. Esses CIs (circuitos integrados) permitirão que você controle oito LEDs separados, utilizando apenas três pinos do Arduino. Mas espere, ainda tem mais: no projeto 18, você controlará 16 LEDs, novamente utilizando apenas três pinos do Arduino.

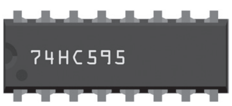
Para demonstrar como os dados são emitidos por esses chips, você criará dois contadores binários, primeiro utilizando um único registrador de deslocamento e, depois, avançando para dois chips em cascata (você aprenderá sobre chips em cascata no projeto 18). O capítulo 6 abordará alguns tópicos bem avançados, por isso, talvez você queira preparar um drinque forte, antes de avançar.

Projeto 17 – Registrador de deslocamento, usado como contador binário de 8 bits

Neste projeto, você utilizará CIs (circuitos integrados) adicionais na forma de registradores de deslocamento, para fazer com que os LEDs contem de forma binária (explicarei o que significa binário em breve). Mais especificamente, você controlará oito LEDs de forma independente, utilizando apenas três pinos de saída do Arduino.

Componentes necessários

1 CI registrador de deslocamento 74HC595



8 resistores de 220 Ω*



8 LEDs de 5 mm



* Ou o equivalente adequado

Conectando os componentes

Analise o diagrama cuidadosamente. Conecte os 3,3 V e o terra nos barramentos inferiores apropriados da protoboard e, em seguida, estenda-os para os barramentos superiores. O chip tem um pequeno sulco em uma de suas pontas; esse sulco vai para a esquerda. O pino 1 está sob o sulco; o pino 8, no canto inferior direito; o pino 9, no canto superior direito e o pino 16 no canto superior esquerdo.

Você necessita de fios que vão da alimentação de 3,3 V para os pinos 10 e 16, e fios do terra para os pinos digitais 8 e 13. Um fio vai do pino digital 8 para o pino digital 12 no CI. Outro vai do pino digital 12 para o pino 14 no CI e, finalmente, um vai do pino digital 11 para o pino 11 no CI.

Os oito LEDs têm um resistor de 220 Ω entre o cátodo e o terra, e o ânodo do LED 1 vai para o pino 15. O ânodo dos LEDs 2 a 8 vão para os pinos 1 a 7 do CI.

Assim que tudo tiver sido conectado, verifique novamente sua fiação, assegurando-se de que o CI e os LEDs estejam conectados corretamente.

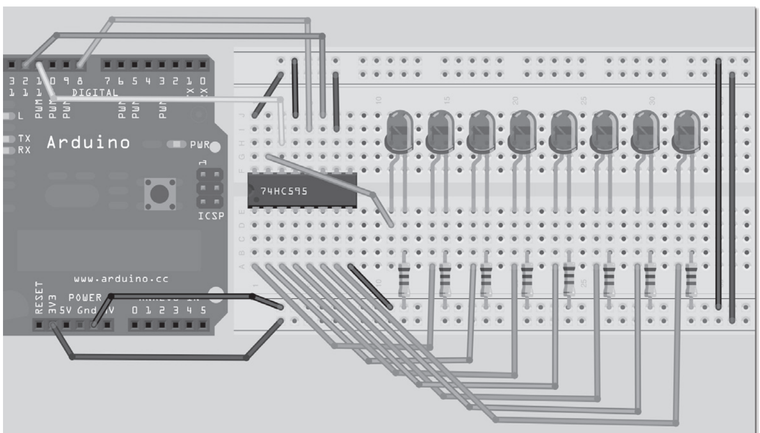


Figura 6.1 – Circuito para o Projeto 17 – Registrador de deslocamento, usado como contador binário de 8 bits (consulte o site da Novatec para versão colorida).

Digite o código

Digite o código da listagem 6.1 e faça seu upload para o Arduino. Assim que o código for executado, você verá os LEDs acendendo e apagando individualmente conforme contam, em números binários, todos os segundos de 0 a 255, reiniciando em seguida.

Listagem 6.1 – Código para o projeto 17

```
// Projeto 17

int latchPin = 8;   // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;  // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;   // Pino conectado ao pino 14 do 74HC595 (Data)
```

```
void setup() {
  // define os pinos como saída
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // conta de 0 a 255
  for (int i = 0; i < 256; i++) {
    // define latchPin como LOW, para permitir o fluxo de dados
    digitalWrite(latchPin, LOW);
    shiftOut(i);
    // define latchPin como HIGH, para fechar e enviar os dados
    digitalWrite(latchPin, HIGH);
    delay(1000);
  }
}

void shiftOut(byte dataOut) {
  // Desloca 8 bits, com o bit menos significativo (LSB) sendo deslocado primeiro, no extremo
  // ascendente do clock
  boolean pinState;
  digitalWrite(dataPin, LOW); // deixa o registrador de deslocamento pronto para enviar dados
  digitalWrite(clockPin, LOW);

  for (int i=0; i<=7; i++)  { // para cada bit em dataOut, envie um bit
    digitalWrite(clockPin, LOW);   // define clockPin como LOW antes de enviar o bit

    // se o valor de DataOut e (E lógico) de uma máscara de bits
    // forem verdadeiros, defina pinState como 1 (HIGH)
    if ( dataOut & (1<<i) ) {
      pinState = HIGH;
    }
    else {
      pinState = LOW;
    }

    // define dataPin como HIGH ou LOW, dependendo de pinState
    digitalWrite(dataPin, pinState);    // envia o bit no extremo ascendente do clock
    digitalWrite(clockPin, HIGH);
  }
  digitalWrite(clockPin, LOW); // interrompa o deslocamento de dados
}
```

Sistema de números binários

Antes de analisar o código e o hardware do projeto 17, vamos nos voltar para o sistema de números binários. É essencial que você compreenda como esse sistema funciona, para que seja capaz de programar um microcontrolador.

Seres humanos utilizam um sistema numérico de base 10, ou decimal, pois temos dez dedos em nossas mãos. Computadores não têm dedos, por isso a melhor forma para um computador contar é utilizando seu equivalente, que é um estado de ON ou OFF (ligado ou desligado; 1 ou 0). Um dispositivo lógico, como um computador, pode detectar se uma voltagem está presente (1) ou não (0), e, portanto, utiliza um sistema numérico binário, ou base 2, pois esse sistema pode ser facilmente representado em um circuito eletrônico, utilizando um estado de voltagem alto ou baixo.

Em nosso sistema numérico de base 10, temos dez dígitos, que vão de 0 a 9. Quando contamos o próximo número depois do 9, o dígito retorna a 0, mas um número 1 é acrescentado à coluna das dezenas à esquerda. Quando a coluna das dezenas atinge 9 e incrementamos esse valor em 1, ela retorna a 0, mas um número 1 é adicionado à coluna das centenas à sua esquerda, e assim por diante:

```
000,001,002,003,004,005,006,007,008,009
010,011,012,013,014,015,016,017,018,019
020,021,023 ………
```

No sistema binário acontece exatamente o mesmo processo, exceto que o número mais alto é 1; por isso, quando adicionamos 1 ao 1, o dígito retorna para 0, e um número 1 é adicionado à coluna da esquerda:

```
000, 001
010, 011
100, 101...
```

Um número de 8 bits (ou 1 byte) é representado como na tabela 6.1.

Tabela 6.1 – Número binário de 8 bits

2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
0	1	0	0	1	0	1	1

O número que mostramos em binário é 1001011, e em decimal é 75.

A conversão foi feita da seguinte maneira:

1 x 1 = 1

1 x 2 = 2

1 x 8 = 8

1 x 64 = 64

Some tudo isso e você terá 75. A tabela 6.2 mostra outros exemplos.

Tabela 6.2 – Exemplos de números binários

Dec	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
75	0	1	0	0	1	0	1	1
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
12	0	0	0	0	1	1	0	0
27	0	0	0	1	1	0	1	1
100	0	1	1	0	0	1	0	0
127	0	1	1	1	1	1	1	1
255	1	1	1	1	1	1	1	1

... e assim por diante.

Dica: Você pode utilizar o Google para converter números decimais em binários, e vice-versa. Assim, para converter 171 de decimal para binário, digite "171 in binary" na caixa de busca do Google, que retornará:

171 = 0b10101011

O prefixo 0b mostra que o número é um número binário, e não um decimal. Assim, a resposta é 10101011.

Para converter um número binário em decimal, faça o inverso. Digite "0b11001100 in decimal" na caixa de busca e ela retornará 0b11001100 = 204.

Agora que você compreende o sistema de números binários, vamos analisar o hardware, antes de analisar o código.

Projeto 17 – Registrador de deslocamento, usado como contador binário de 8 bits – Análise do hardware

Você está utilizando um registrador de deslocamento, mais especificamente o 74HC595. Esse tipo de registrador de deslocamento é um registrador de 8 bits, com entrada serial, saída serial ou paralela, e *latches* (travas) de saída. Isso significa que você pode enviar dados para o registrador de deslocamento em série, e enviar dados para fora em paralelo. Em série significa um bit de cada vez. Em paralelo significa muitos bits (nesse caso, oito) de cada vez.

Os dados entram quando o LATCH está definido como `LOW` (isso permite que dados sejam enviados ao chip), e saem quando o LATCH está definido como `HIGH`. Assim,

você transfere os dados (na forma de 1s e 0s) ao registrador de deslocamento, um bit de cada vez, e então envia para fora oito bits ao mesmo tempo. Cada bit é deslocado conforme o bit seguinte entra. Se um nono bit entrar antes que o latch seja definido como HIGH, o primeiro bit será deslocado para o fim da fila, perdendo-se para sempre.

Registradores de deslocamento são geralmente utilizados na conversão de dados de formato serial para paralelo. Nesse caso, os dados emitidos são 0s e 1s (ou 0 V e 3,3 V), por isso você pode utilizá-los para acender e apagar um conjunto de oito LEDs.

O registrador de deslocamento para este projeto requer apenas três entradas do Arduino. As saídas do Arduino e as entradas do 595 podem ser vistas na tabela 6.3.

Tabela 6.3 – Pinos utilizados

Pino do Arduino	Pino do 595	Descrição
8	12	Entrada do clock do registrador de armazenamento
11	14	Entrada dos dados seriais
12	11	Entrada do clock do registrador de deslocamento

Vamos nos referir ao pino 12 como o pino do clock, ao pino 14 como o pino dos dados e ao pino 11 como o pino do latch.

Imagine o latch como um portão que permite aos dados saírem do 595. Quando o portão está abaixado (LOW), os dados no 595 não podem escapar, mas outros dados podem entrar. Quando o portão está levantado (HIGH), dados não podem mais entrar, mas os dados no registrador de deslocamento são liberados para os oito pinos (QA-QH, ou de Q0 a Q7, dependendo de seu datasheet; consulte a figura 6.2). O clock é simplesmente um pulso de 0s e 1s, e o pino dos dados é por onde você envia dados do Arduino para o 595.

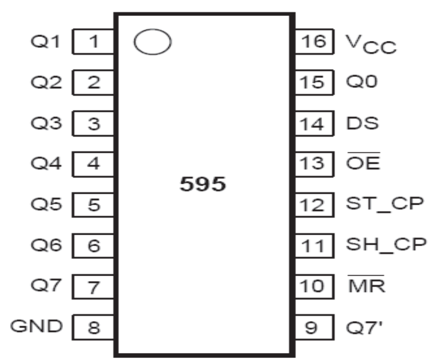


Figura 6.2 – Diagrama de pinos de um chip 595.

Para utilizar o registrador de deslocamento, o pino do latch e o pino do clock devem estar definidos como `LOW`. O pino do latch permanecerá `LOW` até que todos os oito bits tenham sido definidos. Isso permite que os dados entrem no registrador de

armazenamento (simplesmente um local no CI para armazenamento de 1 ou 0). Na sequência, você apresenta um sinal `HIGH` ou `LOW` no pino de dados, e define o pino do clock como `HIGH`. Ao fazê-lo, você armazena no registrador de armazenamento os dados apresentados no pino de dados. Feito isso, você define novamente o clock como `LOW`, e apresenta no pino de dados o próximo bit de dados. Fazendo isso oito vezes, você terá enviado um número completo de 8 bits ao 595. O pino do latch agora será levantado, o que transferirá os dados do registrador de armazenamento para o registrador de deslocamento, fazendo sua saída de Q0 (pino 15) a Q7 (pinos de 1 a 7).

Essa sequência de eventos está descrita na tabela 6.4.

Tabela 6.4 – Sequência de eventos

Pino	Estado	Descrição
Latch	LOW	Latch abaixado, para permitir que os dados entrem
Dados	HIGH	Primeiro bit de dado (1)
Clock	HIGH	Clock vai para HIGH. Dados armazenados
Clock	LOW	Pronto para o próximo bit. Impede novos dados
Dados	HIGH	2° bit de dado (1)
Clock	HIGH	2° bit armazenado
...	...	...
Dados	LOW	8° bit de dado (0)
Clock	HIGH	8° bit armazenado
Clock	LOW	Impeça que novos dados sejam armazenados
Latch	HIGH	Envie os 8 bits em paralelo

Conectei um Analisador Lógico (dispositivo que permite que você veja os 1s e 0s que saem de um dispositivo digital) ao meu 595, enquanto esse programa estava sendo utilizado; a figura 6.3 mostra a saída. A partir desse diagrama, você pode ver que o número binário 00110111 (visto da direita para a esquerda, na figura), ou 55 decimal, foi enviado ao CI.

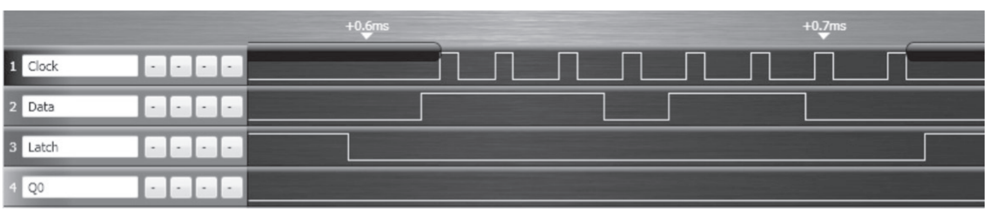


Figura 6.3 – Saída do 595 mostrada em um analisador lógico.

Assim, para resumir o uso que fizemos de um único registrador de deslocamento neste projeto, você tem oito LEDs conectados às oito saídas do registrador. O latch é definido como `LOW`, para habilitar a entrada de dados. Dados são enviados ao pino de dados, um bit por vez, e o pino do clock é definido como `HIGH` para armazenar os

dados. Depois, esse pino é definido como `LOW` novamente, para aguardar o próximo bit. Depois da entrada de todos os oito bits, o latch é definido como `HIGH`, impedindo a entrada de novos dados e definindo o estado dos oito pinos de saída como alto (3,3 V) ou baixo (0 V), dependendo do estado do registrador.

Se você deseja ler mais sobre registradores de deslocamento, consulte o número serial de seu CI (por exemplo, 74HC595N ou SN74HC595N etc.) e faça uma pesquisa em um mecanismo de busca, para encontrar o datasheet específico.

Sou um grande fã do chip 595, pois ele é muito versátil e pode, claramente, elevar o número de pinos de saída digital à sua disposição. O Arduino tem por padrão 19 saídas digitais (os seis pinos analógicos também podem ser utilizados como pinos digitais, numerados de 14 a 19). Utilizando registradores de deslocamento de 8 bits, você pode expandir esse número para 49 (seis vezes o número de pinos do 595, mais um pino que sobra). Além disso, esse chip também opera muito rapidamente, tipicamente a 100 MHz, o que significa que, se quiser, você pode enviar dados a aproximadamente 100 milhões de vezes por segundo (se o Arduino for capaz de fazê-lo). Isso significa que você também pode enviar sinais PWM via software para os CIs, e habilitar controle de brilho para os LEDs.

Como a saída é simplesmente o estado ON e OFF de uma voltagem de saída, ela também pode ser utilizada para ligar e desligar outros dispositivos de baixa potência (ou até mesmo dispositivos de alta potência, utilizando transistores ou relés), ou enviar dados para outros dispositivos (como uma velha impressora de matriz de pontos ou outro dispositivo serial).

Note que, mesmo registradores de deslocamento 595, de fabricantes diferentes, são praticamente idênticos. Também há outros registradores de deslocamento com 16 ou mais saídas. Alguns CIs anunciados como chips para controle de LEDs são, como percebemos ao analisar seus datasheets, simplesmente registradores de deslocamento maiores (por exemplo, o M5450 e o M5451 da STMicrolectronics).

Projeto 17 – Registrador de deslocamento, usado como contador binário de 8 bits – Análise do código

O código para o projeto 17 talvez lhe pareça assustador a princípio. Mas quando dividido em seus componentes, você pode perceber que ele não é tão complexo.

Primeiramente, três variáveis são inicializadas para os três pinos que você utilizará:

```
int latchPin = 8;
int clockPin = 12;
int dataPin = 11;
```

Depois, na inicialização, os pinos são todos definidos como saídas:

```
pinMode(latchPin, OUTPUT);
pinMode(clockPin, OUTPUT);
pinMode(dataPin, OUTPUT);
```

O loop principal simplesmente executa um loop `for`, contando de 0 a 255. Em cada iteração do loop, o `latchPin` é definido como `LOW`, para permitir a entrada de dados. Na sequência, chamamos a função `shiftOut`, passando o valor de `i`, do loop `for`, para a função. Depois, `latchPin` é definido como `HIGH`, impedindo a entrada de mais dados e definindo as saídas dos oito pinos. Por fim, há um intervalo de meio segundo, antes que se inicie a próxima iteração do loop:

```
void loop() {
  // conta de 0 a 255
  for (int i = 0; i < 256; i++) {
    // define latchPin como LOW, para permitir o fluxo de dados
    digitalWrite(latchPin, LOW);
    shiftOut(i);
    // define latchPin como HIGH, para travar e enviar os dados
    digitalWrite(latchPin, HIGH);
    delay(500);
  }
}
```

A função `shiftOut` recebe como parâmetro um byte (número de 8 bits), que será seu número entre 0 e 255. Você escolheu um byte para esse caso, pois ele tem exatamente 8 bits de comprimento, e você tem de enviar apenas oito bits ao registrador de deslocamento:

```
void shiftOut(byte dataOut) {
```

Depois, inicializamos uma variável booleana, `pinState`, que armazenará o estado no qual você deseja que o pino relevante esteja, quando os dados forem enviados (1 ou 0):

```
boolean pinState;
```

Os pinos de dados e do clock são definidos como `LOW` para limpar as linhas de dados e do clock, deixando-as prontas para o envio de novos dados:

```
digitalWrite(dataPin, LOW);
digitalWrite(clockPin, LOW);
```

Depois disso, você está pronto para enviar os oito bits em série para o registrador 595, um bit de cada vez. Um loop `for` que itera oito vezes é preparado:

```
for (int i=0; i<=7; i++)  {
```

O pino do clock é definido como `LOW`, antes do envio de um bit de dado:

```
digitalWrite(clockPin, LOW);
```

Agora, uma instrução `if/else` determina se a variável `pinState` deve ser definida como 1 ou 0:

```
if ( dataOut & (1<<i) ) {
  pinState = HIGH;
}
else {
  pinState = LOW;
}
```

A condição para a instrução `if` é:

```
dataOut & (1<<i).
```

Isso é um exemplo do que chamamos de *máscara de bits*, e agora você está utilizando operadores bit a bit, que são operadores lógicos semelhantes aos operadores booleanos encontrados em projetos anteriores. Entretanto, os operadores bit a bit atuam sobre um número no nível dos bits.

Nesse caso, você está utilizando o operador bit a bit E (`&`) para executar uma operação lógica em dois números. O primeiro número é `dataOut`, e o segundo é o resultado de `(1<<i)`. Antes de continuar, vamos aprender mais sobre os operadores bit a bit.

Operadores bit a bit

Operadores bit a bit realizam cálculos em variáveis no nível dos bits. Há seis operadores bit a bit usuais:

`&`	operador bit a bit E
`|`	operador bit a bit OU
`^`	operador bit a bit XOU
`~`	operador bit a bit NÃO
`<<`	deslocamento de bits para a esquerda
`>>`	deslocamento de bits para a direita

Operadores bit a bit podem ser utilizados apenas entre inteiros. Cada operador realiza um cálculo com base em um conjunto de regras lógicas. Vamos analisar primeiro o operador bit a bit E (`&`), e depois os outros operadores.

E (&) bit a bit

O operador E atua de acordo com esta regra:

Se ambas as entradas forem 1, as saídas resultantes serão 1; do contrário, a saída será 0.

Outra forma de compreender essa regra é:

0 0 1 1 Operando1

0 1 0 1 Operando2

----

0 0 0 1 (Operando1 & Operando2)

Um tipo `int` é um valor de 16 bits, assim, o uso de `&` entre duas expressões `int` provoca 16 operações E simultâneas. Em uma seção de código como esta, temos:

```
int x = 77;     // binário: 0000000001001101
int y = 121;    // binário: 0000000001111001
int z = x & y;  // resultado: 0000000001001001
```

Ou neste caso:

```
77 & 121 = 73
```

OU (|) bit a bit

Se qualquer uma ou ambas as entradas forem 1, o resultado será 1; do contrário, será 0.

0 0 1 1 Operando1

0 1 0 1 Operando2

----

0 1 1 1 (Operando1 | Operando2)

XOU bit a bit (^)

Se apenas uma das entradas for 1, então a saída será 1. Se ambas as entradas forem 1, então a saída será 0.

0 0 1 1 Operando1

0 1 0 1 Operando2

----

0 1 1 0 (Operando1 ^ Operando2)

NÃO bit a bit (~)

O operador bit a bit NÃO é aplicado a um único operando à sua direita.

A saída se torna o oposto da entrada.

0 0 1 1 Operando1

----

1 1 0 0 ~Operando1

Deslocamento de bits para a esquerda (<<), deslocamento de bits para a direita (>>)

Os operadores de deslocamento de bits movimentam todos os bits do valor inteiro para a esquerda ou direita, pelo número de bits especificado no operando da direita.

variável << número_de_bits

Por exemplo:

```
byte x = 9 ;      // binário: 00001001
byte y = x << 3;  // binário: 01001000 (ou 72 decimal)
```

Todos os bits deslocados para fora do final da linha estarão perdidos para sempre. Você pode utilizar o deslocamento à esquerda para multiplicar um número por potências de 2, e o deslocamento à direita para dividir por potências de 2 (experimente para ver o resultado).

Agora que você aprendeu sobre operadores de deslocamento de bits, vamos retornar ao código.

Projeto 17 – Análise do código (continuação)

A condição da instrução `if/else` era:

```
dataOut & (1 << i)
```

Você sabe que essa é uma operação bit a bit E (`&`). O operando da direita, dentro dos parênteses, é uma operação de deslocamento de bits para a esquerda. Isso é uma máscara de bits. O 74HC595 aceitará dados apenas um bit de cada vez. Dessa forma, você tem de converter o número de 8 bits, em `dataOut`, em um número de um único bit, representando cada um dos oito bits, um de cada vez. A máscara de bits permite a você garantir que a variável `pinState` seja definida como `1` ou `0`, dependendo do resultado do cálculo da máscara. O operando à direita é o número 1, deslocado `i` vezes. Como o loop `for` faz com que o valor de `i` vá de 0 a 7, você pode ver que o deslocamento de 1 por `i` vezes (cada vez que acontece o loop), resultará nos seguinte números binários (Tabela 6.5).

Tabela 6.5 – Resultados de 1<<i

Valor de i	Resultado de (1<<i) em número binário
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

Assim, você pode notar que o 1 se movimenta da direita para esquerda, como resultado da operação.

Sabemos que a regra do operador E afirma que:

> Se ambas as entradas forem 1, as saídas resultantes serão 1; do contrário, a saída será 0.

Assim, a condição de

```
dataOut & (1<<i)
```

resultará em 1, se o bit correspondente no mesmo local da máscara de bits for 1; do contrário, ela será zero. Por exemplo, se o valor de `dataOut` for o número decimal `139`, ou o binário `b10001011`, então cada iteração pelo loop resultará nos valores mostrados na tabela 6.6.

Tabela 6.6 – Resultados de b10001011 & (1<<i)

Valor de i	Resultado de b10001011 & (1<<i) em binário
0	00000001
1	00000010
2	00000000
3	00001000
4	00000000
5	00000000
6	00000000
7	10000000

Assim, toda vez que há `1` na posição `i` (lendo da direita para a esquerda), o valor resulta em `1` (ou `HIGH`), e toda vez que há `0` na posição `i`, o valor resulta em `0` (ou `LOW`).

Dessa forma, a condição `if` executará o código de seu bloco se o valor for maior que 0 (ou, em outras palavras, se o bit nessa posição for 1), do contrário (se o bit nessa posição for 0), o bloco `else` executará seu código.

Assim, analisando novamente a instrução `if/else`:

```
if ( dataOut & (1<<i) ) {
  pinState = HIGH;
}
else {
  pinState = LOW;
}
```

E, comparando-a com a tabela-verdade que temos na tabela 6.6, podemos ver que para cada bit no valor de `dataOut` que tem o valor de `1`, o `pinState` correspondente será definido como `HIGH`, e para cada valor de `0`, será definido como `LOW`.

O próximo trecho de código escreve um estado `HIGH` ou `LOW` para o pino de dados, e define o pino do clock como `HIGH`, para escrever esse valor no registro de armazenamento:

```
digitalWrite(dataPin, pinState);
digitalWrite(clockPin, HIGH);
```

Por fim, o pino do clock é definido como `LOW`, para garantir que não sejam escritos outros bits:

```
digitalWrite(clockPin, LOW);
```

Na prática, essa seção de código analisa cada um dos oito bits do valor em `dataOut`, um de cada vez, e define de acordo o pino de dados como `HIGH` ou `LOW`, escrevendo, depois, esse valor no registro de armazenamento.

Isso está simplesmente enviando o número de 8 bits para o 595, um bit de cada vez. Depois, o loop principal define o pino do latch como `HIGH`, para enviar esses oito bits simultaneamente aos pinos 15 e de 1 a 7 (de QA a QH) do registrador de deslocamento. O resultado é que seus oito LEDs exibirão uma representação visual do número binário armazenado no registrador de deslocamento.

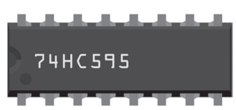
Seu cérebro pode estar cansado depois deste projeto, por isso faça uma pausa, estique as pernas, e prepare outro drinque antes de mergulhar no projeto 18, no qual você utilizará dois registradores de deslocamento encadeados.

Projeto 18 – Contador binário de 8 bits duplo

No projeto 18, você encadeará (ou colocará em cascata) outro 74HC595 ao que você utilizou no projeto 17, para criar um contador binário duplo.

Componentes necessários

2 CI registradores de deslocamento 74HC595



16 resistores limitadores de corrente



8 LEDs vermelhos



8 LEDs verdes



Conectando os componentes

O primeiro 595 utiliza a mesma fiação do projeto 17. O segundo 595 tem os fios de +3,3 V e do terra indo para os mesmos pinos utilizados pelo primeiro 595. Depois, adicione um fio indo do pino 9 do CI 1 para o pino 14 do CI 2, outro fio indo do pino 11 do CI 1 para o pino 11 do CI 2, e um terceiro fio indo do pino 12 do CI 1 para o pino 12 do CI 2.

As mesmas saídas do primeiro 595, indo para o primeiro conjunto de LEDs, vão do segundo CI para o segundo conjunto de LEDs.

Examine cuidadosamente os diagramas das figuras 6.4 e 6.5.

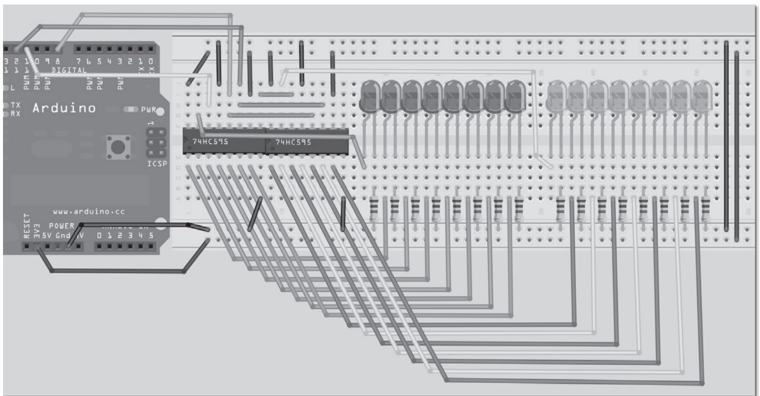


Figura 6.4 – Circuito para o Projeto 18 (consulte o site da Novatec para versão colorida).

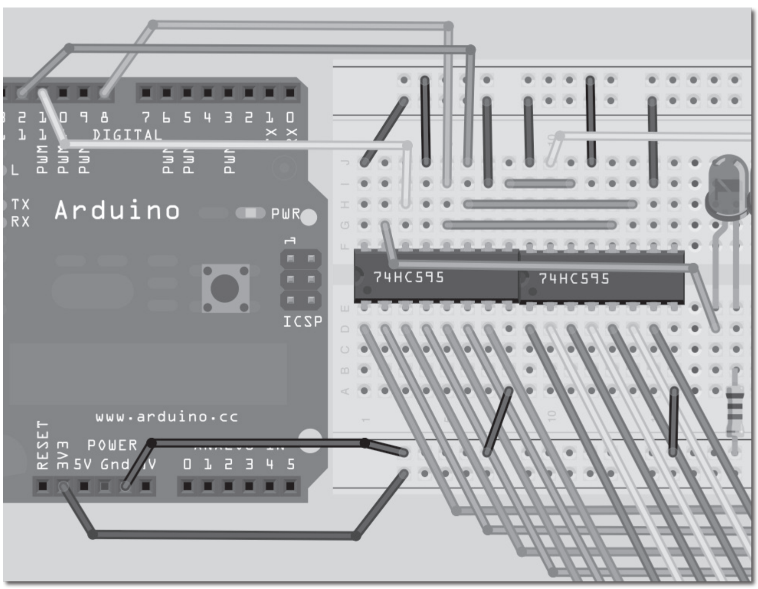


Figura 6.5 – Detalhe da fiação dos CIs para o Projeto 18 (consulte o site da Novatec para versão colorida).

Digite o código

Digite o código da listagem 6.2 e faça seu upload para o Arduino. Quando executá-lo, você verá o conjunto de LEDs verdes contar (em números binários) de 0 a 255, ao mesmo tempo em que o conjunto de LEDs vermelhos conta regressivamente de 255 a 0.

Listagem 6.2 – Código para o projeto 18

```
// Projeto 18

int latchPin = 8;    // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;   // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;    // Pino conectado ao pino 14 do 74HC595 (Data)

void setup() {
  // define os pinos como saída
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  for (int i = 0; i < 256; i++) { //conta de 0 a 255
    digitalWrite(latchPin, LOW); //define latchPin como LOW, para permitir o fluxo de dados
    shiftOut(i);
    shiftOut(255-i);
    // define latchPin como HIGH, para fechar e enviar os dados
    digitalWrite(latchPin, HIGH);
    delay(250 );
  }
}

void shiftOut(byte dataOut) {
  boolean pinState;    // Desloque 8 bits, com o bit menos significativos (LSB) sendo deslocado
                       // primeiro, no extremo ascendente do clock

  digitalWrite(dataPin, LOW);     // libera o registrador de deslocamento, deixando-o pronto
                                  // para enviar dados
  digitalWrite(clockPin, LOW);

  for (int i=0; i<=7; i++)  {     // para cada bit em dataOut, envie um bit
    digitalWrite(clockPin, LOW); // define clockPin como LOW, antes de enviar o bit

    // se o valor de dataOut e (E lógico) uma máscara de bits forem verdadeiros, defina
    // pinState como 1 (HIGH)
    if ( dataOut & (1<<i) ) {
      pinState = HIGH;
    }
```

```
      else {
        pinState = LOW;
      }

      // define dataPin como HIGH ou LOW, dependendo de pinState
      digitalWrite(dataPin, pinState);

      digitalWrite(clockPin, HIGH);    // envia o bit no extremo ascendente do clock
      digitalWrite(dataPin, LOW);
    }

    digitalWrite(clockPin, LOW);   // interrompe o deslocamento
  }
```

Projeto 18 – Análise do código e do hardware

O código para o projeto 18 é idêntico ao do projeto 17, exceto pela adição de

```
shiftOut(255-i);
```

no loop principal. A rotina `shiftOut` envia oito bits para o 595. No loop principal, você utiliza duas chamadas a `shiftOut`, uma enviando o valor de `i`, e outra enviando o valor de `255-i`. Você chama `shiftOut` duas vezes, antes de definir o latch como `HIGH`. Isso enviará dois conjuntos de oito bits, ou 16 bits no total, para os chips 595, antes que o latch seja definido como `HIGH`, para impedir outras operações de escrita nos registradores e para efetuar a saída do conteúdo do registrador de deslocamento aos pinos de saída, acendendo e apagando os LEDs.

O segundo 595 tem a mesma fiação do primeiro. Os pinos do clock e do latch estão ligados aos pinos do primeiro 595. Entretanto, você tem um fio indo do pino 9 do CI 1 para o pino 14 do CI 2. O pino 9 é o pino de saída de dados e o pino 14 é o pino de entrada de dados.

Os dados vindos do Arduino entram no pino 14 do primeiro CI. O segundo chip 595 está encadeado ao primeiro, utilizando o pino 9 do CI 1, que emite dados para o pino 14 do segundo CI, a entrada de dados.

Conforme você faz a entrada de um nono bit, e de outros bits, os dados do CI 1 são deslocados, saindo por seu pino de dados e entrando pelo pino de dados do segundo CI. Dessa forma, quando todos os 16 bits tiverem sido enviados pela conexão de dados a partir do Arduino, os primeiros oito bits enviados terão sido deslocados do primeiro chip para o segundo. Com isso, o segundo chip 595 armazenará os PRIMEIROS oito bits enviados, e o primeiro chip 595, os bits de 9 a 16.

Um número praticamente ilimitado de chips 595 pode ser encadeado dessa forma.

EXERCÍCIO

Utilizando o mesmo circuito do projeto 18 e todos os 16 LEDs, recrie o efeito de luz do seriado “A Super Máquina”, ou dos Cylons de “Battlestar Galactica”, fazendo com que as luzes percorram, acendendo e apagando, os 16 LEDs.

Resumo

O capítulo 6 abordou muitos tópicos referentes ao uso de um CI externo, para fornecer pinos adicionais de saída ao Arduino. Mesmo que você pudesse ter realizado esses projetos sem um CI externo, os registradores de deslocamento facilitaram muito sua vida. Se não os tivesse utilizado, o código para estes projetos seria muito mais complexo. O uso de um CI projetado para receber dados seriais e emitir de modo paralelo é ideal para controlar sequências de LEDs dessa forma.

Nunca tenha receio em utilizar CIs externos. Uma leitura calma e metódica de seus datasheets revelará como eles funcionam. Datasheets podem, a princípio, parecer complicados, mas, uma vez removidos os dados irrelevantes, é fácil compreender como o chip funciona.

No capítulo 7, você continuará utilizando registradores de deslocamento, mas, dessa vez, controlará displays LEDs de matriz de pontos, contendo ao menos 64 LEDs por unidade. Mostrarei como controlar um grande número de LEDs simultaneamente, utilizando uma incrível técnica conhecida como multiplexação.

Assuntos e conceitos abordados no capítulo 6:

- o sistema de números binários e como convertê-los de/para decimais;
- como utilizar um registrador de deslocamento para entrada de dados seriais, e saída de dados em paralelo;
- como utilizar um CI externo para diminuir a complexidade de um projeto;
- como enviar um parâmetro para uma chamada de função;
- o uso de variáveis de tipo booleano;
- o conceito e uso de operadores bit a bit;
- como utilizar operadores bit a bit para criar uma máscara de bits;
- como encadear dois ou mais registradores de deslocamento.

CAPÍTULO 7

Displays de LED

Até aqui, você trabalhou com LEDs individuais de 5 mm. LEDs também podem ser comprados em pacotes, ou como um display de matriz de pontos, sendo que a opção mais popular é uma matriz de 8 x 8 LEDs, ou de 64 LEDs no total. Você também pode obter displays de matriz de pontos bicolores (por exemplo, vermelho e verde) ou até mesmo um display de matriz de pontos RGB, capaz de reproduzir qualquer cor, e com um total de 192 LEDs em um único pacote. Neste capítulo, você trabalhará com um display de matriz de pontos usual, 8 x 8 e de cor única, e aprenderá como exibir imagens e texto. Iniciaremos com uma simples demonstração de como criar uma imagem animada em um display 8 x 8, e depois avançaremos para projetos mais complexos. Nesse processo, você aprenderá um conceito muito importante: a multiplexação.

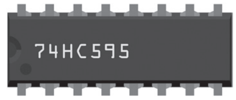
Projeto 19 – Display de matriz de pontos LED – Animação básica

Neste projeto, você utilizará novamente dois registradores de deslocamento, que nesse caso estarão conectados às linhas e colunas do display de matriz de pontos. Depois, você reproduzirá um objeto simples, ou sprite, no display e também o animará. O objetivo principal deste projeto é mostrar-lhe como funciona um display de matriz de pontos e apresentar o conceito da multiplexação, pois essa é uma habilidade valiosíssima.

Componentes necessários

Você necessitará de dois registradores de deslocamento (74HC595) e de oito resistores limitadores de corrente. Também deverá obter um display de matriz de pontos de ânodo comum (C+), assim como suas especificações para que saiba quais pinos devem ser conectados às linhas e colunas.

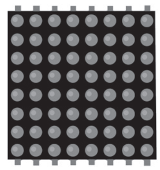
2 CIs registradores de deslocamento 74HC595



8 resistores limitadores de corrente



Display de matriz de pontos 8 x 8 (C+)



Conectando os componentes

Analise o diagrama cuidadosamente. É importante que você não conecte o Arduino ao cabo USB ou à força até que o circuito esteja completo; do contrário, você pode danificar os registradores de deslocamento, ou o display de matriz de pontos. Este é um exercício de fiação complicada, por isso tenha cuidado. Certifique-se de conectar os componentes com calma e de forma metódica.

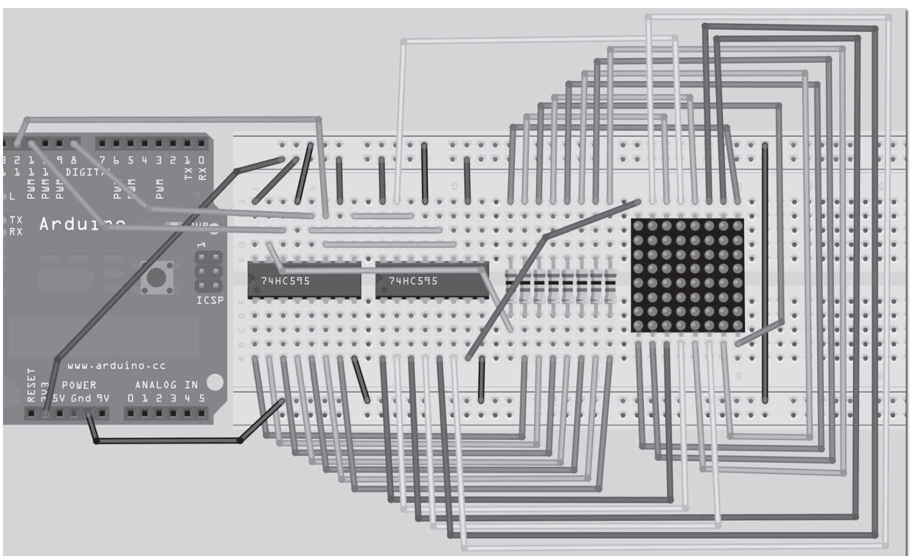


Figura 7.1 – Circuito para o Projeto 19 – Display de matriz de pontos LED – Animação básica (consulte o site da Novatec para versão colorida).

O diagrama da fiação na figura 7.1 é relevante à unidade de matriz de pontos específica utilizada na criação deste projeto, uma miniunidade 8 x 8 de display de matriz de pontos vermelhos. Entretanto, seu display pode ter (e provavelmente terá) pinos

diferentes dos utilizados nesse caso. É *necessário* que você leia o datasheet da unidade que comprou, para garantir que os pinos de saída do registrador de deslocamento sejam conectados corretamente aos pinos do display de pontos. Para um bom tutorial em PDF sobre como ler um datasheet, acesse *www.egr.msu.edu/classes/ece480/goodman/read_datasheet.pdf*.

Para facilitar, a tabela 7.1 mostra quais pinos devem ser conectados no registrador de deslocamento e no display de matriz de pontos. Ajuste o circuito de acordo com o tipo de display que você adquiriu.

Tabela 7.1 – Pinos necessários para o display de matriz de pontos

	Registrador de deslocamento 1	**Registrador de deslocamento 2**
Linha 1	Pino 15	
Linha 2	Pino 1	
Linha 3	Pino 2	
Linha 4	Pino 3	
Linha 5	Pino 4	
Linha 6	Pino 5	
Linha 7	Pino 6	
Linha 8	Pino 7	
Coluna 1		Pino 15
Coluna 2		Pino 1
Coluna 3		Pino 2
Coluna 4		Pino 3
Coluna 5		Pino 4
Coluna 6		Pino 5
Coluna 7		Pino 6
Coluna 8		Pino 7

O diagrama esquemático para o display de matriz de pontos utilizado neste projeto pode ser visto na figura 7.2. Como você pode perceber, as linhas e colunas (ânodos e cátodos) não estão ordenadas logicamente. Utilizando a tabela 7.1 e o diagrama da figura 7.2, você pode ver que o pino 15 no registrador de deslocamento 1 deve ser conectado à linha 1 no display e, portanto, vai (por meio de um resistor) ao pino 9 do display. O pino 1 no registrador de deslocamento deve ir para a linha 2 e, portanto, vai para o pino 14 do display, e assim por diante.

Será necessário que você leia o datasheet do display que adquiriu, e que realize um processo semelhante ao que mostramos para verificar quais pinos do registrador de deslocamento devem ser conectados aos pinos do display LED.

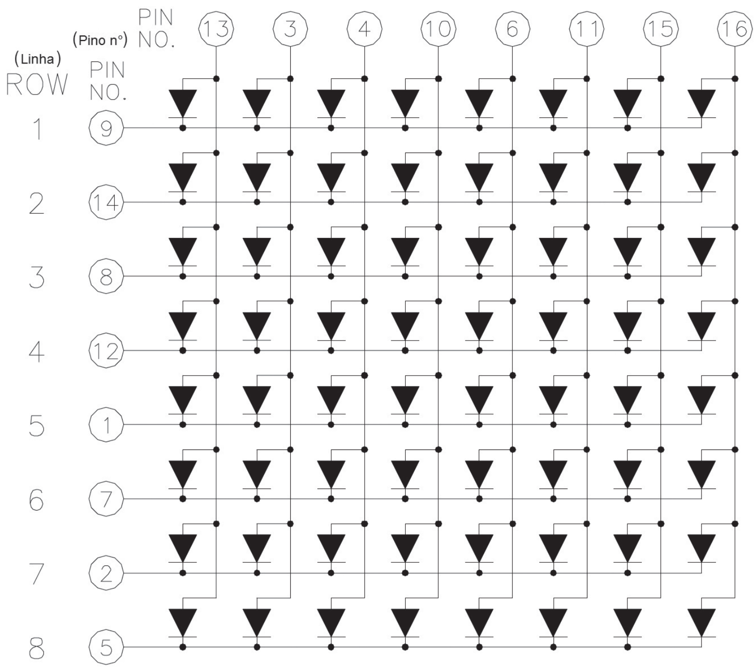


Figura 7.2 – Diagrama esquemático típico para um display de matriz de pontos LED 8 x 8.

Digite o código

Assim que você tiver confirmado que sua fiação está correta, digite o código da listagem 7.1 e faça seu upload para o Arduino. Você também terá de fazer o download da biblioteca TimerOne, que pode ser encontrada no site do Arduino, em *www.arduino.cc/playground/Code/Timer1*. Depois do download da biblioteca, descompacte-a e coloque a pasta `TimerOne` inteira na pasta `hardware/libraries`, dentro da instalação do Arduino. Esse é um exemplo de uma biblioteca externa. O IDE do Arduino vem pré-carregado com muitas bibliotecas, como Ethernet, LiquidCrystal, Servo etc. A biblioteca TimerOne é uma biblioteca externa, e basta fazer seu download e sua instalação para que ela funcione (você terá de reiniciar seu IDE antes que ela seja reconhecida).

Uma biblioteca é simplesmente uma coleção de código escrito por outra pessoa, oferecendo uma funcionalidade que, do contrário, você teria de criar do zero. Tal prática representa o princípio da reutilização de código, e ajuda a acelerar seu processo de desenvolvimento. Afinal, não há nada a ser ganho com a reinvenção da roda. Se alguém já criou um trecho de código que realiza uma tarefa da qual você necessita, e esse código é de domínio público, vá em frente e utilize-o.

Assim que o código tiver sido executado, você verá um coração no display. A cada meio segundo, aproximadamente, o display inverterá para oferecer um efeito de animação básica à imagem.

Listagem 7.1 – Código para o projeto 19

```
// Projeto 19
#include <TimerOne.h>

int latchPin = 8;    // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;   // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;    // Pino conectado ao pino 14 do 74HC595 (Data)

byte led[8];         // array de bytes com 8 elementos para armazenar o sprite

void setup() {
  pinMode(latchPin, OUTPUT);   // define os 3 pinos digitais como saída
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  led[0] = B11111111;    // insere a representação binária da imagem
  led[1] = B10000001;    // no array
  led[2] = B10111101;
  led[3] = B10100101;
  led[4] = B10100101;
  led[5] = B10111101;
  led[6] = B10000001;
  led[7] = B11111111;
  // define um timer com duração de 10000 microssegundos (1/100 de um segundo)
  Timer1.initialize(10000);
  // anexa a função screenUpdate ao timer de interrupção
  Timer1.attachInterrupt(screenUpdate);
}

void loop() {
  for (int i=0; i<8; i++) {
    led[i]= ~led[i];  // inverte cada linha da imagem binária
  }
  delay(500);
}

void screenUpdate() {             // função para exibir a imagem
  byte row = B10000000;           // linha 1
  for (byte k = 0; k < 9; k++) {
    digitalWrite(latchPin, LOW);   // abre o latch, deixando-o pronto para receber dados
  shiftIt(~led[k]);            // envia o array de LEDs (invertido) para os chips
  shiftIt(row);                // envia o número binário da linha para os chips

  // Fecha o latch, enviando os dados no registrador para o display de matriz
  digitalWrite(latchPin, HIGH);
  row = row >> 1;   // deslocamento para a direita
  }
}
```

```
void shiftIt(byte dataOut) {       // Desloca 8 bits, com o menos significativo deslocado
                                  // primeiro, durante o extremo ascendente do clock

  boolean pinState;
  digitalWrite(dataPin, LOW);     // libera o registrador de deslocamento, deixando-o pronto
                                  // para enviar dados

  for (int i=0; i<8; i++)  {      // para cada bit em dataOut, envie um bit
    digitalWrite(clockPin, LOW); // define clockPin como LOW, antes de enviar o bit

    // se o valor de dataOut e (E lógico) uma máscara de bits
    // forem verdadeiros, defina pinState como 1 (HIGH)
    if ( dataOut & (1<<i) ) {
      pinState = HIGH;
    }
    else {
      pinState = LOW;
    }
    // define dataPin como HIGH ou LOW, dependendo de pinState
    digitalWrite(dataPin, pinState);
    digitalWrite(clockPin, HIGH);     // envia o bit durante o extremo ascendente do clock
    digitalWrite(dataPin, LOW);
  }
digitalWrite(clockPin, LOW); // interrompe o deslocamento
}
```

Projeto 19 – Display de matriz de pontos LED – Animação básica – Análise do hardware

Para este projeto, veremos como funciona o hardware, antes do código. Isso fará com que seja mais fácil compreender o código posteriormente.

Você aprendeu como utilizar o 74HC595 nos projetos anteriores. A única adição ao circuito, dessa vez, é uma unidade de display 8 x 8 de pontos LED.

Unidades de matriz de pontos tipicamente vêm no formato de uma matriz de LEDs de 5 x 7 ou 8 x 8. Os LEDs são conectados à matriz de forma que o ânodo ou o cátodo de cada LED seja comum em cada linha. Em outras palavras, em uma unidade de matriz de pontos LED habitual, cada linha de LEDs terá todos os seus ânodos conectados. Os cátodos de cada coluna também estarão todos conectados. A razão disso vai ficar aparente muito em breve.

Uma típica unidade de matriz de pontos 8 x 8 colorida terá 16 pinos, oito para as linhas e oito para as colunas. Você também pode obter unidades bicolores (por exemplo, nas cores vermelho e verde), assim como unidades RGB — aquelas utilizadas

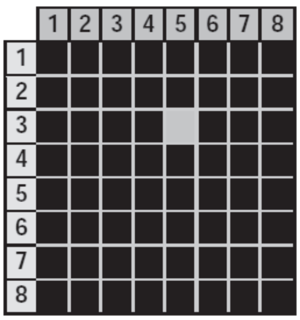
em grandes telões. Unidades bicolores ou tricolores (RGB) têm dois ou três LEDs em cada pixel do array. São LEDs muito pequenos, e que ficam posicionados bem próximos uns dos outros.

Ao acender combinações diferentes de vermelho, verde ou azul em cada pixel, além de variar seu brilho, pode-se obter qualquer cor.

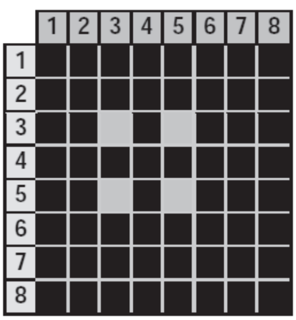
O motivo de as linhas e colunas serem todas conectadas é para minimizar o número de pinos necessários. Se não o fizéssemos, uma única unidade de matriz de pontos 8 x 8 colorida necessitaria de 65 pinos, um para cada LED e um conector de ânodo ou cátodo comum. Ligando a fiação das linhas e colunas, apenas 16 pinos são necessários.

Entretanto, isso representa um problema se você deseja que um LED específico acenda em certa posição. Caso, por exemplo, você tivesse uma unidade de ânodo comum e quisesse acender o LED na posição X, Y, de valores 5, 3 (quinta coluna, terceira linha), você aplicaria uma corrente à terceira linha e o terra ao pino da quinta coluna.

O LED na quinta coluna e na terceira linha acenderia.



Agora, imagine que você deseja acender também o LED na coluna 3, linha 5. Então, você aplicaria uma corrente à quinta linha e o terra ao pino da terceira coluna. O LED correspondente agora seria iluminado. Mas espere... os LEDs da coluna 3, linha 3 e coluna 5, linha 5 também acenderam.



Isso ocorre porque você está aplicando energia às linhas 3 e 5, e o terra às colunas 3 e 5. Você não pode apagar os LEDs indesejados sem apagar também aqueles que devem estar acesos. Parece não haver forma de acender apenas os LEDs necessários, com a fiação que utilizamos para linhas e colunas. A única maneira de isso dar certo seria se tivéssemos uma pinagem separada para cada LED, o que faria o número de pinos pular de 16 para 65. Uma unidade de matriz de pontos de 65 pinos teria uma fiação muito complexa e seria muito difícil de controlar, pois você necessitaria de um microcontrolador de ao menos 64 saídas digitais.

Haveria uma solução para esse problema? Sim, ela existe, e é o que chamamos de *multiplexação* (*multiplexing*, ou *muxing*).

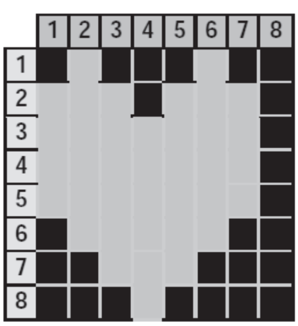
Multiplexação

A multiplexação é a técnica de acender uma linha do display de cada vez. Selecionando a coluna que contém a linha, que, por sua vez, contém o LED que você deseja acender, e ligando a alimentação para essa linha (ou da forma oposta, para displays comuns de cátodo), os LEDs escolhidos nessa linha serão iluminados. Essa linha será, então, apagada, e a próxima acesa, novamente com as colunas apropriadas escolhidas, e fazendo com que os LEDs da segunda linha agora sejam iluminados. Repita esse processo para cada linha até que você atinja a base e, então, reinicie do topo.

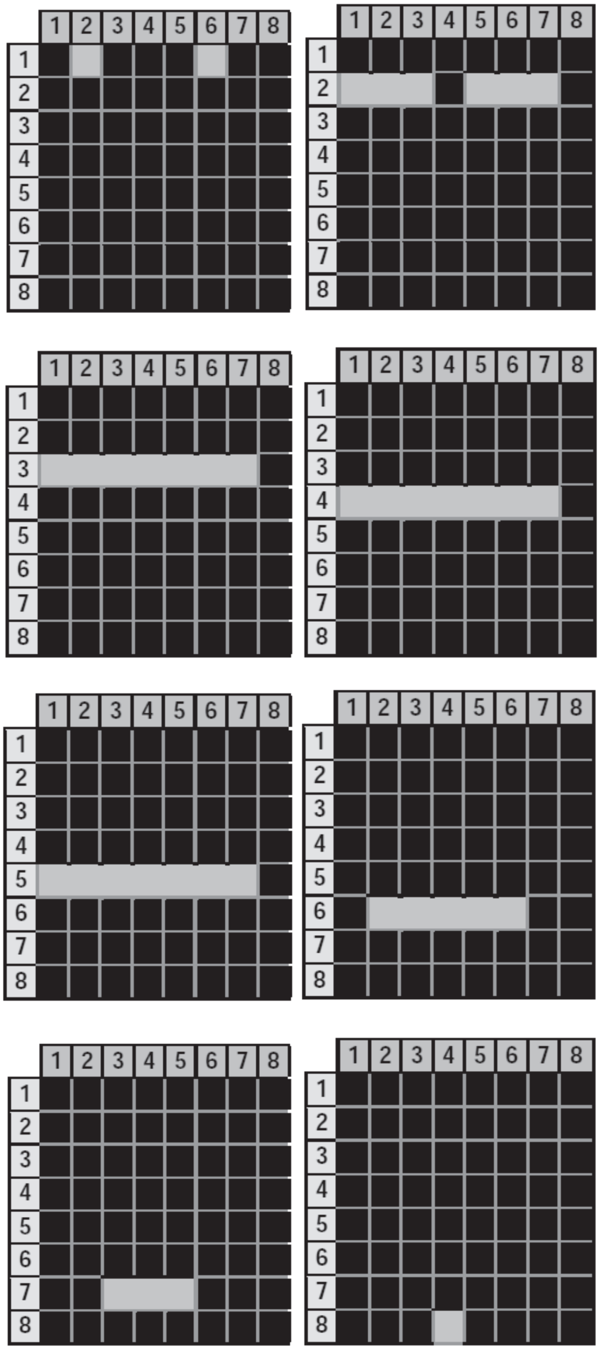
Se isso for feito a uma velocidade suficiente (mais de 100 Hz, ou cem vezes por segundo), o fenômeno de *persistência da visão* (em que uma pós-imagem permanece na retina por, aproximadamente, 1/25 de um segundo) fará com que o display pareça acender por inteiro, mesmo que cada linha acenda e apague em sequência.

Utilizando essa técnica, você soluciona o problema da exibição de LEDs individuais, sem que outros LEDs na mesma coluna também acendam.

Por exemplo, vamos supor que você queira exibir a imagem a seguir em seu display:



Cada linha seria acesa da seguinte maneira:



Descendo pelas linhas, iluminando os LEDs respectivos em cada coluna, e fazendo isso muito rapidamente (a mais de 100 Hz), o olho humano reconhecerá a imagem por inteiro, e um coração poderá ser reconhecido no padrão dos LEDs.

Você está utilizando essa técnica de multiplexação no código do projeto. É dessa forma que você exibe a animação do coração, sem acender LEDs desnecessários.

Projeto 19 – Display de matriz de pontos LED – Animação básica – Análise do código

O código para este projeto utiliza um recurso do chip ATmega, conhecido como Hardware Timer: essencialmente, um timer que pode ser utilizado para disparar um evento. Em seu caso, você está definindo a ISR (Interrupt Service Routine, ou rotina de serviço de interrupção) para que dispare a cada 10 mil microssegundos, o equivalente a um centésimo de segundo.

Neste código, você utiliza uma biblioteca que facilita o uso de interrupções, a TimerOne, e que torna muito fácil criar uma ISR. Você simplesmente diz à função qual é o intervalo (nesse caso, 10 mil microssegundos), e passa o nome da função que deseja ativar cada vez que a interrupção for disparada (nesse caso, a função `screenUpdate()`)

TimerOne é uma biblioteca externa, por isso você tem de incluí-la em seu código. Isso pode ser feito com facilidade utilizando a diretiva `#include`:

```
#include <TimerOne.h>
```

Na sequência, são declarados os pinos utilizados para interfacear os registradores de deslocamento:

```
int latchPin = 8;   // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;  // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;   // Pino conectado ao pino 14 do 74HC595 (Data)
```

Depois, você cria um array de tipo byte, com oito elementos. O array `led[8]` será utilizado para armazenar a imagem a ser exibida no display de matriz de pontos:

```
byte led[8];    // array de bytes com 8 elementos para armazenar o sprite
```

Na rotina de setup, você define os pinos do latch, do clock e de dados como saídas:

```
void setup() {
  pinMode(latchPin, OUTPUT);   // define os 3 pinos digitais como saída
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
```

Assim que os pinos tiverem sido definidos como saída, o array `led` será carregado com as imagens binárias de 8 bits que serão exibidas em cada linha do display de matriz de pontos 8 x 8:

```
  led[0] = B11111111;    // insere a representação binária da imagem
  led[1] = B10000001;    // no array
  led[2] = B10111101;
  led[3] = B10100101;
  led[4] = B10100101;
```

```
led[5] = B10111101;
led[6] = B10000001;
led[7] = B11111111;
```

Analisando o array anterior, você pode imaginar a imagem que será exibida, que é uma caixa dentro de uma caixa. Os 1s indicam os LEDs que acenderão, e os 0s indicam os LEDs que ficarão apagados. Você pode, evidentemente, ajustar os 1s e 0s como quiser, e criar um sprite 8 x 8 de sua escolha.

Depois disso, utilizamos a função `Timer1`. Primeiramente, ela tem de ser inicializada com a frequência na qual será ativada. Nesse caso, você define o período como 10 mil microssegundos, ou um centésimo de segundo. Assim que a interrupção tiver sido inicializada, você deve anexar a ela uma função que será executada sempre que o período de tempo for atingido. Esta é a função `screenUpdate()`, que disparará a cada centésimo de segundo:

```
// define um timer com duração de 10000 microssegundos (1/100 de segundo)
Timer1.initialize(10000);
// anexa a função screenUpdate ao timer de interrupção
Timer1.attachInterrupt(screenUpdate);
```

No loop principal, um loop `for` percorre cada um dos oito elementos do array `led` e inverte o conteúdo, utilizando o operador bit a bit ~, ou NÃO. Isso simplesmente transforma a imagem binária em um negativo dela mesma, transformando todos os 1s em 0s e todos os 0s em 1s. Então, esperamos 500 milissegundos antes de repetir o processo.

```
for (int i=0; i<8; i++) {
  led[i]= ~led[i];  // inverte cada linha da imagem binária
}
delay(500);
```

Agora temos a função `screenUpdate()`, ativada pela interrupção a cada centésimo de segundo. Toda essa rotina é muito importante, pois garante que os LEDs do array na matriz de pontos acendam corretamente e exibam a imagem que você deseja representar. Trata-se de uma função muito simples, mas eficiente.

```
void screenUpdate() {            // função para exibir a imagem
  byte row = B10000000;          // linha 1
  for (byte k = 0; k < 9; k++) {
    digitalWrite(latchPin, LOW);   // abre o latch, deixando-o pronto para receber dados
  shiftIt(~led[k] );           // envia o array de LEDs (invertido) para os chips
  shiftIt(row );               // envia o número binário da linha para os chips

  // Fecha o latch, enviando os dados no registrador para o display de matriz
  digitalWrite(latchPin, HIGH);
  row = row >> 1; // deslocamento para a direita
  }
}
```

Um byte, `row`, é declarado e inicializado com o valor B10000000:

```
byte row = B10000000;    // linha 1
```

Você, agora, percorre o array `led`, e envia esses dados para o registrador de deslocamento (processado com o operador bit a bit NÃO (~), para garantir que as colunas que você deseja exibir estejam apagadas, ou ligadas ao terra), seguidos pela linha:

```
for (byte k = 0; k < 9; k++) {
  digitalWrite(latchPin, LOW); // abre o latch, deixando-o pronto para receber dados
  shiftIt(~led[k] );           // envia o array de LEDs (invertido) para os chips
  shiftIt(row );               // envia o número binário da linha para os chips
```

Assim que você tiver deslocado os oito bits da linha atual, o valor em `row` será deslocado um bit para a direita, de forma que a próxima linha seja exibida (por exemplo, a linha com o 1 é exibida). Você aprendeu sobre o comando `bitshift` no capítulo 6.

```
row = row >> 1;   // deslocamento para a direita
```

Lembre-se de que vimos, na análise do hardware, que a rotina de multiplexação está exibindo apenas uma linha de cada vez e, depois, apagando essa linha e exibindo a linha seguinte. Isso é feito a 100 Hz, rápido demais para que o olho humano perceba uma cintilação.

Por fim, você tem a função `shiftIt`, a mesma dos projetos anteriores com base em registradores de deslocamento, a qual envia os dados para os chips do 74HC595:

```
void shiftIt(byte dataOut)
```

Portanto, o conceito básico deste projeto é uma rotina de interrupção executada a cada centésimo de segundo. Nessa rotina, você apenas analisa o conteúdo de um array de buffer de tela (nesse caso, `led[]`), exibindo-o na unidade de matriz de pontos, uma linha de cada vez. Isso é feito tão rapidamente que, para o olho humano, tudo parece acontecer ao mesmo tempo.

O loop principal do programa está simplesmente alterando o conteúdo do array do buffer de tela, permitindo que a ISR faça o trabalho necessário.

A animação neste projeto é muito simples, mas manipulando os 1s e 0s no buffer, você pode representar o que quiser na unidade de matriz de pontos, desde formas diferentes até textos de rolagem horizontal. No próximo projeto, vamos experimentar uma variação do que fizemos, na qual você criará um sprite animado com rolagem horizontal.

Projeto 20 – Display de matriz de pontos LED – Sprite com rolagem horizontal

Neste projeto, você utilizará o mesmo circuito do projeto anterior, mas com uma pequena variação no código para criar uma animação de vários quadros (frames), que também se movimentará da direita para a esquerda. Nesse processo, você será

apresentado ao conceito de arrays multidimensionais, e também aprenderá um pequeno truque para realizar a rotação de bits (ou deslocamento circular). Para iniciar, você utilizará exatamente o mesmo circuito do projeto 19.

Digite o código

Digite e faça o upload do código da listagem 7.2.

Listagem 7.2 – Código para o projeto 20

```
// Projeto 20
#include <TimerOne.h>

int latchPin = 8;    // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;   // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;    // Pino conectado ao pino 14 do 74HC595 (Data)
byte frame = 0;      // variável para armazenar o quadro atual sendo exibido

byte led[8][8] ={ {0, 56, 92, 158, 158, 130, 68, 56},    // 8 quadros de uma animação
           {0, 56, 124, 186, 146, 130, 68, 56},
           {0, 56, 116, 242, 242, 130, 68, 56},
           {0, 56, 68, 226, 242, 226, 68, 56},
           {0, 56, 68, 130, 242, 242, 116, 56},
           {0, 56, 68, 130, 146, 186, 124, 56},
           {0, 56, 68, 130, 158, 158, 92, 56},
           {0, 56, 68, 142, 158, 142, 68, 56} };

void setup() {
  pinMode(latchPin, OUTPUT);   // define os 3 pinos digitais como saídas
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  Timer1.initialize(10000);  // define um timer com duração de 10000 microssegundos
  Timer1.attachInterrupt(screenUpdate);    // anexa a função screenUpdate
}

void loop() {
  for (int i=0; i<8; i++) {  // faz um loop, percorrendo todos os 8 quadros da animação
    for (int j=0; j<8; j++) {  // faz um loop pelas 8 linhas de cada quadro
      led[i][j] = led[i][j] << 1 | led[i][j] >> 7; // rotação bit a bit
    }
  }
  frame++;               // vai para o próximo quadro da animação
  if (frame>7) { frame =0; }     // certifica-se de retornar ao frame 0, depois de passar do 7
  delay(100);              // espera um pouco entre cada frame
}
```

```
void screenUpdate() {            // função para exibir a imagem
  byte row = B10000000;         // linha 1
  for (byte k = 0; k < 9; k++) {
    digitalWrite(latchPin, LOW); // abre o latch, deixando-o pronto para receber dados

    shiftIt(~led[frame][k]); // envia o array de LEDs (invertido) para os chips
    shiftIt(row);        // envia o número binário da linha para os chips

    // Fecha o latch, enviando os dados nos registradores para a matriz de pontos
    digitalWrite(latchPin, HIGH);
    row = row >> 1; // deslocamento para a direita
  }
}

void shiftIt(byte dataOut) {
  // Desloca 8 bits, com o menos significativo deslocado primeiro, durante o extremo ascendente
  // do clock
  boolean pinState;

  // libera o registrador de deslocamento, deixando-o pronto para enviar dados
  digitalWrite(dataPin, LOW);

  // para cada bit em dataOut, envie um bit
  for (int i=0; i<8; i++)  {
    // define clockPin como LOW, antes de enviar o bit
    digitalWrite(clockPin, LOW);

    // se o valor de dataOut e (E lógico) uma máscara de bits
    // forem verdadeiros, define pinState como 1 (HIGH)
    if ( dataOut & (1<<i) ) {
      pinState = HIGH;
    }
    else {
      pinState = LOW;
    }

    // define dataPin como HIGH ou LOW, dependendo do pinState
    digitalWrite(dataPin, pinState);
    // envia o bit durante o extremo ascendente do clock
    digitalWrite(clockPin, HIGH);
    digitalWrite(dataPin, LOW);
  }

digitalWrite(clockPin, LOW);    // interrompe o deslocamento
}
```

Quando você executar o projeto 20, poderá ver uma roda animada, rolando horizontalmente. O hardware não sofreu alterações, por isso não é necessário discuti-lo. Vamos descobrir como funciona o código.

Projeto 20 – Display de matriz de pontos LED – Sprite com rolagem horizontal – Análise do código

Novamente, você carrega a biblioteca TimerOne e define os três pinos que controlam os registradores de deslocamento:

```
#include <TimerOne.h>

int latchPin = 8;   // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;  // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;   // Pino conectado ao pino 14 do 74HC595 (Data)
```

Depois, você declara uma variável de tipo `byte`, inicializando-a como 0. Ela armazenará o número do quadro atualmente exibido na animação de oito quadros:

```
byte frame = 0;   // variável para armazenar o quadro atual sendo exibido
```

Na sequência, você prepara um array bidimensional de tipo `byte`:

```
byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56}, // 8 quadros de uma animação
           {0, 56, 124, 186, 146, 130, 68, 56},
           {0, 56, 116, 242, 242, 130, 68, 56},
           {0, 56, 68, 226, 242, 226, 68, 56},
           {0, 56, 68, 130, 242, 242, 116, 56},
           {0, 56, 68, 130, 146, 186, 124, 56},
           {0, 56, 68, 130, 158, 158, 92, 56},
           {0, 56, 68, 142, 158, 142, 68, 56} };
```

Arrays foram apresentados no capítulo 3. Um array é um conjunto de variáveis que podem ser acessadas utilizando um número de índice. O array que utilizamos agora é diferente, pois tem dois conjuntos de números de índice para os elementos. No capítulo 3, você declarou um array unidimensional desta forma:

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

Aqui, você tem de criar um array bidimensional, com dois conjuntos de números de índice. Nesse caso, seu array tem 8 x 8, ou 64 elementos no total. Um array bidimensional é praticamente idêntico a uma tabela bidimensional: você pode acessar uma célula individual, referenciando os números de linha e de coluna correspondentes. A tabela 7.2 mostra como acessar os elementos em seu array.

As linhas representam o primeiro número do índice do array, por exemplo `byte led[7][..]`, e as colunas representam o segundo índice, por exemplo `byte led[..][7]`. Para acessar o número `158` na linha 6, coluna 4, você utilizaria `byte led[6][4]`.

Tabela 7.2 – Elementos em seu array

	0	1	2	3	4	5	6	7
0	0	56	92	158	158	130	68	56
1	0	56	124	186	146	130	68	56
2	0	56	116	242	242	130	68	56
3	0	56	68	226	242	226	68	56
4	0	56	68	130	242	242	116	56
5	0	56	68	130	146	186	124	56
6	0	56	68	130	158	158	92	56
7	0	56	68	142	158	142	68	56

Note que, ao declarar o array, você também o inicializou com dados. Para inicializar dados em um array bidimensional, você coloca todos eles dentro de chaves globais, e cada conjunto de dados dentro de suas próprias chaves com uma vírgula ao final, da seguinte maneira:

```
byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56},
          {0, 56, 124, 186, 146, 130, 68, 56},
          {0, 56, 116, 242, 242, 130, 68, 56},    // etc., etc.
```

O array bidimensional armazenará os oito quadros de sua animação. O primeiro índice do array fará referência ao quadro da animação, e o segundo a qual das oito linhas de números de 8 bits formará o padrão de LEDs que devem acender e apagar. Para economizar espaço no código, os números foram convertidos de binários para decimais. Se você visse os números binários, poderia discernir a animação da figura 7.3.

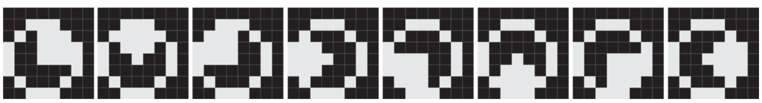


Figura 7.3 – Animação da roda girando.

Evidentemente, você pode alterar essa animação para representar o que quiser, aumentando ou diminuindo o número de quadros. Esboce sua animação no papel, converta as linhas para números binários de 8 bits, e coloque-os em seu array.

No loop de inicialização, você define os três pinos novamente como saída, cria um objeto de timer com duração de 10 mil microssegundos e anexa a função `screenUpdate()` à interrupção:

```
void setup() {
  pinMode(latchPin, OUTPUT);   // define os 3 pinos digitais como saídas
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  Timer1.initialize(10000);         // define um timer com duração de 10000 microssegundos
  Timer1.attachInterrupt(screenUpdate);   // anexa a função screenUpdate
}
```

No loop principal, você faz um loop percorrendo cada uma das oito linhas do sprite, assim como no projeto 19. Entretanto, este loop está dentro de outro loop, que se repete oito vezes e controla qual frame você deseja exibir:

```
void loop() {
  for (int i=0; i<8; i++) {    // faz um loop por todos os 8 quadros da animação
    for (int j=0; j<8; j++) {    // faz um loop pelas 8 linhas em cada quadro
```

Na sequência, você pega todos os elementos do array, um de cada vez, e desloca seus valores uma posição para a esquerda. Entretanto, utilizando um pequeno e interessante truque de lógica, você garante que todos os bits deslocados para fora, no lado esquerdo, retornem no lado direito. Isso é feito com o seguinte comando:

```
led[i][j]= led[i][j] << 1 | led[i][j] >> 7;   // rotação bit a bit
```

Aqui, o elemento atual do array, escolhido pelos inteiros `i` e `j`, é deslocado uma posição para a esquerda. Entretanto, você, depois, pega esse resultado e aplica um OU lógico com o número de valor `led [i][j]` deslocado sete posições para a direita. Vejamos como isso funciona.

Suponha que o valor atual de `led[i][j]` seja `156`, ou o número binário 10011100. Se esse número for deslocado para a esquerda uma posição, você terá 00111000. Agora, se você pegar o mesmo número, `156`, e deslocá-lo para a direita sete vezes, terá como resultado 00000001. Em outras palavras, fazendo isso, você desloca o dígito binário da extrema esquerda para o lado direito. Em seguida, você realiza uma operação lógica bit a bit OU nos dois números. Lembre-se de que o cálculo bit a bit OU produzirá um número 1 se qualquer um dos dígitos for 1, da seguinte maneira:

00111000 |

00000001 =

00111001

Com isso, você deslocou o número uma posição para a esquerda, e utilizou um OU, comparando esse resultado com o mesmo número deslocado sete posições para a direita. Como você pode ver no cálculo, o resultado é o mesmo de deslocar o número para a esquerda uma vez, e deslocar para o lado direito qualquer dígito que tenha saído pelo lado esquerdo. Isso é conhecido como *rotação bit a bit* ou *deslocamento circular*, técnica frequentemente utilizada em criptografia digital. Você pode realizar uma rotação bit a bit em um dígito binário de qualquer extensão, utilizando o seguinte cálculo:

```
i << n | i >> (a - n);
```

No qual `n` é o número de dígitos que você deseja rotacionar e `a` é a extensão, em bits, de seu dígito original.

Na sequência, você incrementa o valor de `frame` em `1`, verifica se ele é maior que `7` e, caso seja, define o número novamente como 0. Isso fará um ciclo por cada um dos oito quadros da animação, um quadro de cada vez, até que atinja o fim dos quadros, e depois repetirá a operação. Por fim, há uma espera de 100 milissegundos.

```
frame++;          // vai para o próximo quadro na animação
if (frame>7) { frame =0;}  // certifica-se de retornar ao frame 0 depois de passar do 7
delay(100);         // espera um pouco entre cada frame
```

Depois, você executa as funções `screenUpdate()` e `shiftIt()`, da mesma forma que nos projetos anteriores, com base em registradores de deslocamento. No próximo projeto, você utilizará novamente uma matriz de pontos LED, mas dessa vez não fará uso de registradores de deslocamento. Em vez disso, você utilizará o popular chip MAX7219.

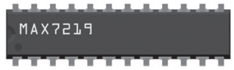
Projeto 21 – Display de matriz de pontos LED – Mensagem com rolagem horizontal

Há muitas formas diferentes de controlar LEDs. Utilizar registradores de deslocamento é apenas uma das opções, e tem suas vantagens. Entretanto, há muitos outros CIs disponíveis, especificamente projetados para controlar displays LED, facilitando muito a sua vida. Um dos CIs controladores de LEDs mais populares na comunidade do Arduino é o MAX7219, um controlador de displays de LED com interface serial de oito dígitos, feito pela Maxim. Esses chips são projetados para controlar displays de LED numéricos de sete segmentos com até oito dígitos, displays de gráficos de barras, ou displays LED de matriz de pontos 8 x 8, sendo este o uso que faremos deles. O IDE do Arduino vem como uma biblioteca, Matrix, que acompanha código de exemplo escrito especificamente para os chips MAX7219. Essa biblioteca facilita, e muito, a utilização desses chips. Entretanto, neste projeto você não utilizará nenhuma biblioteca externa. Em vez disso, você escolherá o caminho mais difícil, escrevendo todo o código você mesmo. Dessa forma, você aprenderá exatamente como o chip MAX7219 funciona, e poderá transferir suas habilidades para o uso de qualquer outro chip controlador de LEDs.

Componentes necessários

Será necessário um CI controlador de LED MAX7219. Como alternativa, você pode utilizar um AS1107 da Austria Microsystems, praticamente idêntico ao MAX7219, e que funcionará sem necessidade de alterações no seu código ou circuito. Dessa vez, o display de matriz de pontos 8 x 8 deve ser do tipo cátodo comum, uma vez que o chip MAX não funcionará com um display de ânodo comum.

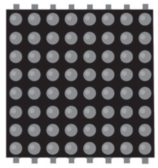
MAX7219 (ou AS1107)



Resistor limitador de corrente



Display de matriz de pontos 8 x 8 (C-)



Conectando os componentes

Analise o diagrama da figura 7.4 cuidadosamente. Certifique-se de que seu Arduino esteja desligado enquanto conecta os fios. A fiação do MAX7219 para o display de matriz de pontos, na Figura 7.4, foi criada de acordo com a unidade de display específica que utilizei. Os pinos de seu display podem ser diferentes. Isso não é relevante, simplesmente conecte os pinos que saem do MAX7219 aos pinos de coluna e linha apropriados em seu display (consulte a tabela 7.3 para a pinagem). Uma leitura horizontal mostrará quais dispositivos estão conectados a quais pinos. No display, as colunas são os cátodos, e as linhas, os ânodos. Em meu display, verifiquei que a linha 1 estivesse na base e a linha 8 no topo. Você pode ter de inverter a ordem em seu display, se notar que as letras estão invertidas ou ao contrário. Conecte os 5 V do Arduino ao barramento positivo da protoboard, e o terra ao barramento do terra.

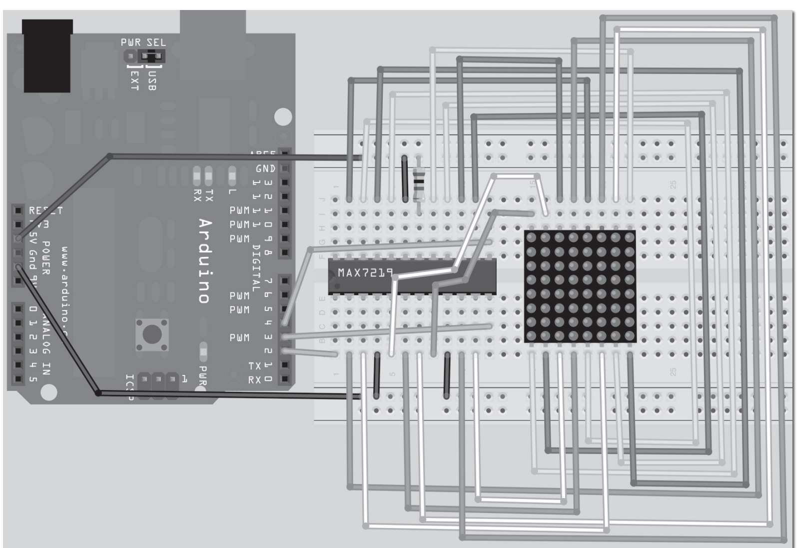


Figura 7.4 – Circuito para o Projeto 21 (consulte o site da Novatec para versão colorida).

Tabela 7.3 – Pinagem entre o Arduino, o CI e o display de matriz de pontos

Arduino	MAX7219	Display	Outros
Digital 2	1 (DIN)		
Digital 3	12 (LOAD)		
Digital 4	13 (CLK)		
	4,9		Gnd (Terra)
	19		+5 V
	18 (ISET)		Resistor para +5 V
	2 (DIG 0)	Coluna 1	
	11 (DIG 1)	Coluna 2	
	6 (DIG 2)	Coluna 3	
	7 (DIG 3)	Coluna 4	
	3 (DIG 4)	Coluna 5	
	10 (DIG 5)	Coluna 6	
	5 (DIG 6)	Coluna 7	
	8 (DIG 7)	Coluna 8	
	22 (SEG DP)	Linha 1	
	14 (SEG A)	Linha 2	
	16 (SEG B)	Linha 3	
	20 (SEG C)	Linha 4	
	23 (SEG D)	Linha 5	
	21 (SEG E)	Linha 6	
	15 (SEG F)	Linha 7	
	17 (SEG G)	Linha 8	

Verifique suas conexões antes de ligar o Arduino.

Digite o código

Digite e faça o upload do código da listagem 7.3.

Listagem 7.3 – Código para o projeto 21

```
#include <avr/pgmspace.h>
#include <TimerOne.h>

int DataPin = 2;  // Pino 1 no MAX
int LoadPin = 3;  // Pino 12 no MAX
int ClockPin = 4; // Pino 13 no MAX
byte buffer[8];

static byte font[][8] PROGMEM = {
```

```
// Apenas os caracteres ASCII imprimíveis (32-126)
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000000, B00000100},
{B00001010, B00001010, B00001010, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00001010, B00011111, B00001010, B00011111, B00001010, B00011111, B00001010},
{B00000111, B00001100, B00010100, B00001100, B00000110, B00000101, B00000110, B00011100},
{B00011001, B00011010, B00000010, B00000100, B00000100, B00001000, B00001011, B00010011},
{B00000110, B00001010, B00010010, B00010100, B00001001, B00010110, B00010110, B00001001},
{B00000100, B00000100, B00000100, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000010, B00000100, B00001000, B00001000, B00001000, B00001000, B00000100, B00000010},
{B00001000, B00000100, B00000010, B00000010, B00000010, B00000010, B00000100, B00001000},
{B00010101, B00001110, B00011111, B00001110, B00010101, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000100, B00000100, B00011111, B00000100, B00000100, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000110, B00000100, B00001000},
{B00000000, B00000000, B00000000, B00000000, B00001110, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000100},
{B00000001, B00000010, B00000010, B00000100, B00000100, B00001000, B00001000, B00010000},
{B00001110, B00010001, B00010011, B00010001, B00010101, B00010001, B00011001, B00001110},
{B00000100, B00001100, B00010100, B00000100, B00000100, B00000100, B00000100, B00011111},
{B00001110, B00010001, B00010001, B00000010, B00000100, B00001000, B00010000, B00011111},
{B00001110, B00010001, B00000001, B00001110, B00000001, B00000001, B00010001, B00001110},
{B00010000, B00010000, B00010100, B00010100, B00011111, B00000100, B00000100, B00000100},
{B00011111, B00010000, B00010000, B00011110, B00000001, B00000001, B00000001, B00011110},
{B00000111, B00001000, B00010000, B00011110, B00010001, B00010001, B00010001, B00001110},
{B00011111, B00000001, B00000001, B00000001, B00000010, B00000100, B00001000, B00010000},
{B00001110, B00010001, B00010001, B00001110, B00010001, B00010001, B00010001, B00001110},
{B00001110, B00010001, B00010001, B00001111, B00000001, B00000001, B00000001, B00000001},
{B00000000, B00000100, B00000100, B00000000, B00000000, B00000100, B00000100, B00000000},
{B00000000, B00000100, B00000100, B00000000, B00000000, B00000100, B00000100, B00001000},
{B00000001, B00000010, B00000100, B00001000, B00001000, B00000100, B00000010, B00000001},
{B00000000, B00000000, B00000000, B00011110, B00000000, B00011110, B00000000, B00000000},
{B00010000, B00001000, B00000100, B00000010, B00000010, B00000100, B00001000, B00010000},
{B00001110, B00010001, B00010001, B00000010, B00000100, B00000100, B00000000, B00000100},
{B00001110, B00010001, B00010001, B00010101, B00010101, B00010001, B00010001, B00011110},
{B00001110, B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001},
{B00011110, B00010001, B00010001, B00011110, B00010001, B00010001, B00010001, B00011110},
{B00000111, B00001000, B00010000, B00010000, B00010000, B00010000, B00001000, B00000111},
{B00011100, B00010010, B00010001, B00010001, B00010001, B00010001, B00010010, B00011100},
{B00011111, B00010000, B00010000, B00011110, B00010000, B00010000, B00010000, B00011111},
{B00011111, B00010000, B00010000, B00011110, B00010000, B00010000, B00010000, B00010000},
{B00001110, B00010001, B00010000, B00010000, B00010111, B00010001, B00010001, B00001110},
{B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001, B00010001},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00011111},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00010100, B00001000},
{B00010001, B00010010, B00010100, B00011000, B00010100, B00010010, B00010001, B00010001},
{B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00011111},
```

```
{B00010001, B00011011, B00011111, B00010101, B00010001, B00010001, B00010001, B00010001},
{B00010001, B00011001, B00011001, B00010101, B00010101, B00010011, B00010011, B00010001},
{B00001110, B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001110},
{B00011110, B00010001, B00010001, B00011110, B00010000, B00010000, B00010000, B00010000},
{B00001110, B00010001, B00010001, B00010001, B00010001, B00010101, B00010011, B00001111},
{B00011110, B00010001, B00010001, B00011110, B00010100, B00010010, B00010001, B00010001},
{B00001110, B00010001, B00010000, B00001000, B00000110, B00000001, B00010001, B00001110},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001110},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001010, B00000100},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010101, B00010101, B00001010},
{B00010001, B00010001, B00001010, B00000100, B00000100, B00001010, B00010001, B00010001},
{B00010001, B00010001, B00001010, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00011111, B00000001, B00000010, B00000100, B00001000, B00010000, B00010000, B00011111},
{B00001110, B00001000, B00001000, B00001000, B00001000, B00001000, B00001000, B00001110},
{B00010000, B00001000, B00001000, B00000100, B00000100, B00000010, B00000010, B00000001},
{B00001110, B00000010, B00000010, B00000010, B00000010, B00000010, B00000010, B00001110},
{B00000100, B00001010, B00010001, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00011111},
{B00001000, B00000100, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00001110, B00010010, B00010010, B00010010, B00001111},
{B00000000, B00010000, B00010000, B00010000, B00011100, B00010010, B00010010, B00011100},
{B00000000, B00000000, B00000000, B00001110, B00010000, B00010000, B00010000, B00001110},
{B00000000, B00000001, B00000001, B00000001, B00000111, B00001001, B00001001, B00000111},
{B00000000, B00000000, B00000000, B00011100, B00010010, B00011110, B00010000, B00001110},
{B00000000, B00000011, B00000100, B00000100, B00000110, B00000100, B00000100, B00000100},
{B00000000, B00001110, B00001010, B00001010, B00001110, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010000, B00011100, B00010010, B00010010, B00010010},
{B00000000, B00000000, B00000100, B00000000, B00000100, B00000100, B00000100, B00000100},
{B00000000, B00000010, B00000000, B00000010, B00000010, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010100, B00011000, B00011000, B00010100, B00010000},
{B00000000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000000, B00001010, B00010101, B00010001, B00010001, B00010001},
{B00000000, B00000000, B00000000, B00010100, B00011010, B00010010, B00010010, B00010010},
{B00000000, B00000000, B00000000, B00001100, B00010010, B00010010, B00010010, B00001100},
{B00000000, B00011100, B00010010, B00010010, B00011100, B00010000, B00010000, B00010000},
{B00000000, B00001110, B00010010, B00010010, B00001110, B00000010, B00000010, B00000001},
{B00000000, B00000000, B00000000, B00001010, B00001100, B00001000, B00001000, B00001000},
{B00000000, B00000000, B00001110, B00010000, B00001000, B00000100, B00000010, B00011110},
{B00000000, B00010000, B00010000, B00011100, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000000, B00010010, B00010010, B00010010, B00010010, B00001100},
{B00000000, B00000000, B00000000, B00010001, B00010001, B00010001, B00001010, B00000100},
{B00000000, B00000000, B00000000, B00010001, B00010001, B00010001, B00010101, B00001010},
{B00000000, B00000000, B00000000, B00010001, B00001010, B00000100, B00001010, B00010001},
{B00000000, B00000000, B00010001, B00001010, B00000100, B00001000, B00001000, B00010000},
{B00000000, B00000000, B00000000, B00011111, B00000010, B00000100, B00001000, B00011111},
```

```
{B00000010, B00000100, B00000100, B00000100, B00001000, B00000100, B00000100, B00000010},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00001000, B00000100, B00000100, B00000100, B00000010, B00000100, B00000100, B00001000},
{B00000000, B00000000, B00000000, B00001010, B00011110, B00010100, B00000000, B00000000}
};

void clearDisplay() {
  for (byte x=0; x<8; x++) {
    buffer[x] = B00000000;
  }
  screenUpdate();
}

void initMAX7219() {
  pinMode(DataPin, OUTPUT);
  pinMode(LoadPin, OUTPUT);
  pinMode(ClockPin, OUTPUT);
  clearDisplay();
  writeData(B00001011, B00000111);    // limite de varredura definido entre 0 e 7
  writeData(B00001001, B00000000);    // modo de decodificação desligado
  writeData(B00001100, B00000001);    // define o registrador de desligamento (shutdown) para a
                                      // operação normal
  intensity(15);               // Apenas valores de 0 a 15 (4 bits)
}

void intensity(int intensity) {
  writeData(B00001010, intensity);    // B00001010 é o Registrador de Intensidade (Intensity Register)
}

void writeData(byte MSB, byte LSB) {
  byte mask;
  digitalWrite(LoadPin, LOW);           // deixa o LoadPin pronto para receber dados
  // Envia o bit mais significativo (most significant byte, ou MSB)
  for (mask = B10000000; mask > 0; mask >>= 1) {  // itera, percorrendo a máscara de bits
    digitalWrite(ClockPin, LOW);
    if (MSB & mask){             // se o E bit a bit for verdadeiro
      digitalWrite(DataPin,HIGH);       // envia 1
    }
    else{                      // se o E bit a bit for falso
      digitalWrite(DataPin,LOW);        // envia 0
    }
    digitalWrite(ClockPin, HIGH);         // clock no estado alto, os dados entram
  }
```

```
  // Envie o bit menos significativo para os dados
  for (mask = B10000000; mask>0; mask >>= 1) {   // itera, percorrendo a máscara de bits
    digitalWrite(ClockPin, LOW);
    if (LSB & mask) {                 // se o E bit a bit for verdadeiro
      digitalWrite(DataPin,HIGH);     // envia 1
    }
    else {                        // se o E bit a bit for falso
      digitalWrite(DataPin,LOW);      // envia 0
    }
    digitalWrite(ClockPin, HIGH);     // clock no estado alto, os dados entram
  }
  digitalWrite(LoadPin, HIGH);   // trava os dados
  digitalWrite(ClockPin, LOW);
}


void scroll(char myString[], int speed) {
  byte firstChrRow, secondChrRow;
  byte ledOutput;
  byte chrPointer = 0;   // inicializa o ponteiro de posição da string
  byte Char1, Char2;     // os dois caracteres que serão exibidos
  byte scrollBit = 0;
  byte strLength = 0;
  unsigned long time;
  unsigned long counter;

  // Incrementa a contagem, até que o final da string seja alcançado
  while (myString[strLength]) {strLength++;}

  counter = millis();

  while (chrPointer < (strLength-1)) {
    time = millis();
    if (time > (counter + speed)) {
      Char1 = myString[chrPointer];
      Char2 = myString[chrPointer+1];
      for (byte y= 0; y<8; y++) {
        firstChrRow = pgm_read_byte(&font[Char1 - 32][y]);
        secondChrRow = (pgm_read_byte(&font[Char2 - 32][y])) << 1;
        ledOutput = (firstChrRow << scrollBit) | (secondChrRow >> (8 - scrollBit) );
        buffer[y] = ledOutput;
      }
      scrollBit++;
      if (scrollBit > 6) {
      scrollBit = 0;
```

```
        chrPointer++;
        }
        counter = millis();
      }
    }
  }

  void screenUpdate() {
    for (byte row = 0; row < 8; row++) {
      writeData(row+1, buffer[row]);
    }
  }

  void setup() {
    initMAX7219();
    Timer1.initialize(10000);   // inicializa timer1 e define o período de interrupção
    Timer1.attachInterrupt(screenUpdate);
  }

  void loop() {
    clearDisplay();
    scroll("   BEGINNING ARDUINO   ", 45);
    scroll("   Chapter 7 - LED Displays   ", 45);
    scroll("   HELLO WORLD!!!  :)   ", 45);
  }
```

Quando você fizer o upload do código, verá uma mensagem de rolagem horizontal no display.

Projeto 21 – Display LED de matriz de pontos – Mensagem com rolagem horizontal – Análise do hardware

Para facilitar a compreensão do código, você primeiro deve saber como funciona o chip MAX7219, por isso, analisaremos o hardware antes do código.

O MAX7219 opera de modo muito semelhante ao dos registradores de deslocamento que vimos antes, no sentido de que você faz a entrada de dados de modo serial, bit a bit. Um total de 16 bits deve ser carregado no dispositivo de cada vez. O chip é de fácil utilização, e usa apenas três pinos do Arduino. O pino digital 2 vai para o pino 1 do MAX, que é o pino de entrada de dados (Data In, ou DIN). O pino digital 3 vai para o pino 12 do MAX, o LOAD, e o pino digital 4 vai para o pino 13 do MAX, o clock (CLK). Consulte a figura 7.5 para a pinagem do MAX7219.

O pino LOAD é colocado no estado baixo, e o primeiro bit de dados é definido como HIGH ou LOW no pino DIN. O pino CLK é definido para oscilar entre LOW e HIGH.

No extremo ascendente do pulso do clock, o bit no pino DIN é deslocado para o registrador interno. Então, o pulso do clock cai para LOW, e o próximo bit de dados é definido no pino DIN antes que o processo se repita. Depois de todos os 16 bits de dados terem sido colocados no registrador, conforme o clock sobe e desce 16 vezes, o pino LOAD será finalmente definido como HIGH, o que travará os dados no registrador.

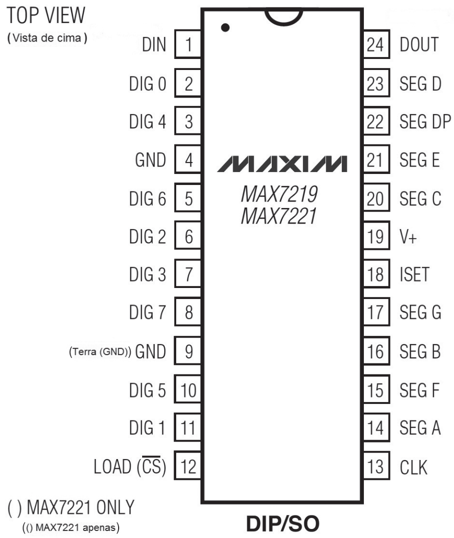


Figura 7.5 – Diagrama de pinos para o MAX7219.

A figura 7.6 é o diagrama de sincronismo do datasheet do MAX7219, e mostra como os três pinos são manipulados para enviar os bits de dados, de D0 a D15, para o dispositivo. O pino DOUT (pino 24) não é utilizado neste projeto. Porém, se você tivesse encadeado mais um chip MAX7219, o DOUT do primeiro chip estaria conectado ao DIN do segundo, e assim por diante. Dados saem do pino DOUT durante o extremo descendente do ciclo do clock.

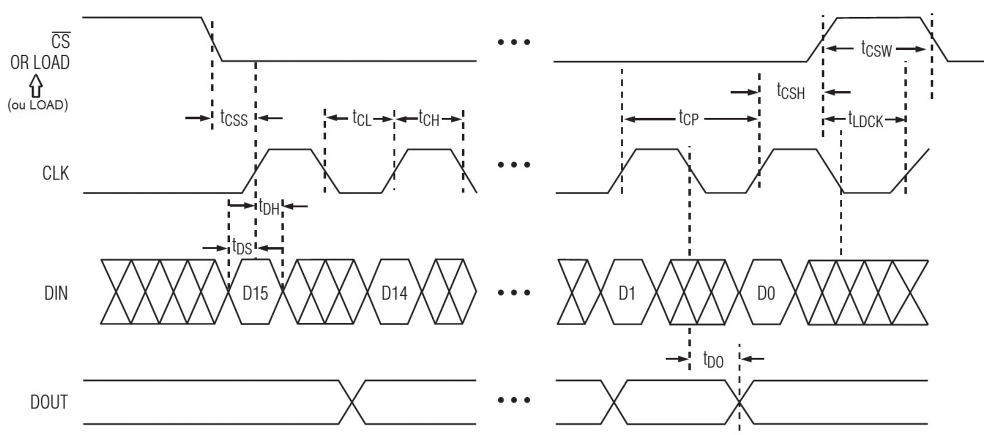


Figura 7.6 – Diagrama de tempo para o MAX7219.

Você tem de recriar essa sequência de sincronismo em seu código, para conseguir enviar os códigos apropriados ao chip. O chip pode receber uma corrente de até 100 mA, mais do que suficiente para a maioria dos displays de matriz de pontos. Caso você queira ler o datasheet do MAX7219, pode fazer seu download no site da Maxim, em *http://datasheets.maxim-ic.com/en/ds/MAX7219-MAX7221.pdf*.

O dispositivo aceita dados em 16 bits. O D15, ou bit mais significativo (*most significant bit*, ou MSB), é enviado primeiro, por isso a ordem decresce de D15 para D0, o bit menos significativo (*least significant bit*, ou LSB). Os primeiros quatro bits são bits "don't care" (sem importância), ou seja, bits que não serão utilizados pelo CI, por isso podem ser qualquer coisa. Os próximos quatro bits representam o endereço do registrador, e os oito bits finais representam os dados. A tabela 7.4 mostra o formato dos dados seriais, e a tabela 7.5 mostra o mapa de endereço dos registradores.

Tabela 7.4 – Formato de dados seriais (16 bits) do MAX7219

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ENDEREÇO					MSB		DADOS X X X X X X X			LSB	

Tabela 7.5 – Mapa de endereço de registradores do MAX7219

Registro	Endereço					Código Hexadecimal
	D15-D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Dígito 0	X	0	0	0	1	0xX1
Dígito 1	X	0	0	1	0	0xX2
Dígito 2	X	0	0	1	1	0xX3
Dígito 3	X	0	1	0	0	0xX4
Dígito 4	X	0	1	0	1	0xX5
Dígito 5	X	0	1	1	0	0xX6
Dígito 6	X	0	1	1	1	0xX7
Dígito 7	X	1	0	0	0	0xX8
Modo de decodificação (Decode Mode)	X	1	0	0	1	0xX9
Intensidade (Intensity)	X	1	0	1	0	0xXA
Limite de varredura (Scan Limit)	X	1	0	1	1	0xXB
Desligamento (Shutdown)	X	1	1	0	0	0xXC
Teste do display (Display test)	X	1	1	1	1	0xXF

Por exemplo, como você pode ver pelo mapa de endereço dos registradores na tabela 7.5, o endereço para o registrador de intensidade é 1010 binário. O registrador de intensidade define o brilho do display com valores que vão do menos intenso, em 0, ao mais intenso, em 15 (B000 a B1111). Para definir a intensidade como 15 (máxima), você enviaria os 16 bits a seguir, com o bit mais significativo (o bit mais à esquerda) sendo enviado primeiro, e o bit menos significativo (o bit mais à direita) sendo enviado por último (ou seja, o número está na ordem inversa dos bits):

```
0000101000001111
```

Os quatro bits menos significativos dos primeiros oito bits têm o valor de B1010, endereço do registrador de intensidade. Os quatro bits mais significativos dos primeiros oito bits são bits "don't care", por isso você envia B0000. Os oito bits seguintes são os dados que estão sendo enviados ao registro. Nesse caso, você deseja enviar o valor B1111 ao registrador de intensidade. Os primeiros quatro bits são novamente do tipo "don't care", por isso você envia B0000. Enviando esses 16 bits ao dispositivo, você define a intensidade do display como máxima. O valor inteiro de 16 bits que você deseja enviar é B000101000001111, mas como são enviados primeiro os MSBs (bits mais significativos) e depois os LSBs (bits menos significativos), o número é enviado em ordem inversa, nesse caso, B111100000101000.

Outro endereço que você utilizará é o do limite de varredura (*scan limit*). Lembre-se de que o MAX7219 é projetado para trabalhar com displays LED de sete segmentos (Figura 7.7).



Figura 7.7 – Display LED de sete segmentos (imagem por Tony Jewell).

O limite de varredura decide quantos dos oito dígitos devem ser acesos. Em seu caso, você não está utilizando displays de sete segmentos, mas displays de matriz de pontos 8 x 8. Os dígitos correspondem às colunas em seu display. Você deseja que todas as oito colunas estejam sempre habilitadas, por isso o registrador do limite de varredura será definido como B00000111 (dígitos de 0 a 7, e 7 em binário é B111).

O registrador do modo de decodificação (*decode mode register*) é relevante apenas se você estiver utilizando displays de sete segmentos, por isso será definido como B00000000, para desligar a decodificação.

Por fim, você definirá o registro de desligamento (*shutdown register*) como B00000001, para garantir que ele esteja em operação normal, e não em modo de desligamento. Se você definir o registro de desligamento como B00000000, todas as fontes de alimentação serão direcionadas para o terra, deixando o display em branco.

Para mais informações sobre o CI MAX7219, leia seu datasheet. Procure as partes do texto relevantes ao seu projeto, e você verá que é muito mais fácil compreendê-las do que parece à primeira vista.

Agora que você (assim esperamos) compreende como o MAX7219 funciona, vamos analisar o código e descobrir como exibir o texto com rolagem horizontal.

Projeto 21 – Display LED de matriz de pontos – Mensagem com rolagem horizontal – Análise do código

Sua primeira ação no início do sketch é carregar as duas bibliotecas que serão utilizadas no código:

```
#include <avr/pgmspace.h>
#include <TimerOne.h>
```

A primeira biblioteca é a `pgmspace`, ou a biblioteca dos utilitários do Program Space, cujas funções permitem ao seu programa acessar dados armazenados em espaço de programa ou na memória flash. O Arduino com chip ATmega328 tem 32 kB de memória flash (2 kB são utilizados pelo bootloader, por isso 30 kB estão disponíveis). O Arduino Mega tem 128 kB de memória flash, 4 kB dos quais são utilizados pelo bootloader. O espaço de programa é exatamente o que indica seu nome: o espaço em que seu programa será armazenado. Você pode utilizar o espaço livre na memória flash empregando os utilitários do Program Space. É nela que você armazenará o extenso array bidimensional com a fonte de seus caracteres.

A segunda biblioteca é a `TimerOne`, utilizada pela primeira vez no projeto 19. Na sequência, declaramos os três pinos digitais que farão interface com o MAX7219:

```
int DataPin = 2;     // Pino 1 no MAX
int LoadPin = 3;     // Pino 12 no MAX
int ClockPin = 4;    // Pino 13 no MAX
```

Depois, você cria um array de tipo byte com oito elementos:

```
byte buffer[8];
```

Esse array armazenará o padrão de bits, que decidirá quais LEDs devem estar acesos ou apagados quando o display estiver ativo.

Depois, temos um extenso array bidimensional de tipo byte:

```
static byte font[][8] PROGMEM = {
// Apenas os caracteres ASCII que podem ser impressos (32-126)
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000000, B00000100},
..etc
```

Esse array está armazenando o padrão de bits que forma a fonte utilizada para exibir o texto no display. Trata-se de um array bidimensional de tipo `static byte`. Depois da declaração do array, você também adicionou o comando `PROGMEM`, uma função dos utilitários do Program Space que diz ao compilador para armazenar esse array na memória flash, em vez de armazenar na SRAM (Static Random Access Memory).

SRAM é o espaço na memória do chip ATmega normalmente utilizado para armazenar as variáveis e strings de caracteres de seu sketch. Quando utilizados, esses dados são copiados do espaço de programa para a SRAM. Entretanto, o array utilizado para armazenar a fonte de texto é formado de 96 caracteres, compostos de oito bits cada. O array tem 96 x 8 elementos, o que corresponde a 768 elementos no total, e cada elemento é um byte (8 bits). A fonte, portanto, ocupa 768 bytes no total. O chip ATmega328 tem apenas 2 kB, ou aproximadamente 2.000 bytes de espaço na memória para variáveis. Quando você soma isso às outras variáveis e strings de texto utilizadas no programa, corre um sério risco de rapidamente ficar sem memória.

O Arduino não tem como lhe avisar de que a memória está acabando. Em vez disso, ele simplesmente para de funcionar. Para impedir que isso ocorra, você armazena esse array na memória flash, em vez de armazenar na SRAM, uma vez que ela tem muito mais espaço disponível. O sketch tem cerca de 2.800 bytes, e o array, pouco menos de 800 bytes, por isso você utilizará algo em torno de 3,6 kB dos 30 kB de memória flash disponíveis.

Em seguida, você cria as diversas funções que serão necessárias para o programa. A primeira simplesmente limpa o display. Quaisquer bits armazenados no array `buffer` serão exibidos na matriz. A função `clearDisplay()` simplesmente percorre todos os oito elementos do array e define seus valores como 0, para que nenhum LED esteja aceso e o display fique em branco. Depois, ela chama a função `screenUpdate()`, que exibe na matriz o padrão armazenado no array `buffer[]`. Nesse caso, como o buffer contém apenas zeros, nada será exibido.

```
void clearDisplay() {
  for (byte x=0; x<8; x++) {
    buffer[x] = B00000000;
  }
  screenUpdate();
}
```

A função seguinte, `initMAX7219()`, prepara o chip MAX7219 para uso. Primeiramente, os três pinos são definidos como `OUTPUT`:

```
void initMAX7219() {
  pinMode(DataPin, OUTPUT);
  pinMode(LoadPin, OUTPUT);
  pinMode(ClockPin, OUTPUT);
```

Depois, limpamos o display:

```
clearDisplay();
```

O limite de varredura é definido como `7` (em binário), o modo de decodificação é desligado e o registrador de desligamento é definido para a operação normal:

```
writeData(B00001011, B00000111);  // limite de varredura definido entre 0 e 7
writeData(B00001001, B00000000); // modo de decodificação desligado
writeData(B00001100, B00000001); // define o registrador de desligamento (shutdown) para a
                                 // operação normal
```

Então, a intensidade é definida como máxima, chamando a função `intensity()`:

```
intensity(15); // Apenas valores de 0 a 15 (4 bits)
```

Em seguida, temos a função `intensity()` em si, que simplesmente pega o valor transmitido a ela e escreve esse dado no registrador de intensidade, chamando a função `writeData()`:

```
void intensity(int intensity) {
  writeData(B00001010, intensity);  // B0001010 é o Registrador de Intensidade (Intensity Register)
}
```

A função seguinte realiza a maioria do trabalho pesado. Seu objetivo é escrever os dados no MAX7219, um bit de cada vez. A função exige dois parâmetros, ambos do tipo `byte`, representando o byte (e não bit) mais significativo e o byte menos significativo do número de 16 bits.

```
void writeData(byte MSB, byte LSB) {
```

Uma variável de tipo `byte`, chamada `mask`, é declarada:

```
byte mask;
```

Ela será utilizada como uma máscara de bits (conceito apresentado no projeto 17), para escolha do bit correto a ser enviado.

Na sequência, o `loadPin` é definido como `LOW`. Isso destrava os dados no registro do CI, deixando-o pronto para receber novos dados:

```
digitalWrite(LoadPin, LOW); // deixa o LoadPin pronto para receber dados
```

Agora, você deve enviar o byte mais significativo do número de 16 bits para o chip, com o bit mais à esquerda (mais significativo) sendo enviado primeiro. Para tanto, você utiliza dois conjuntos de loops `for`, um para o MSB e outro para o LSB. O loop utiliza uma máscara de bits para percorrer todos os oito bits. O uso de uma função bit a bit E (`&`) decide se o bit atual é `1` ou `0`, e define o `dataPin` de acordo, como `HIGH` ou `LOW`. O `clockPin` é definido como `LOW`, e o valor `HIGH` ou `LOW` é escrito no `dataPin`:

```
// Envia o bit mais significativo
for (mask = B10000000; mask>0; mask >>= 1) {   // itera, percorrendo a máscara de bits
  digitalWrite(ClockPin, LOW);
  if (MSB & mask) {                            // se o E bit a bit for verdadeiro
    digitalWrite(DataPin,HIGH);                // envia 1
  }
  else {                                       // se o E bit a bit for falso
    digitalWrite(DataPin,LOW);                 // envia 0
  }
  digitalWrite(ClockPin, HIGH);                // clock no estado alto, os dados entram
}
```

Por fim, o `loadPin` é definido como `HIGH`, para garantir que os 16 bits sejam travados no registro do chip, e o `clockPin` é definido como `LOW`, uma vez que o último pulso havia sido `HIGH` (o clock deve oscilar entre `HIGH` e `LOW`, para que os dados pulsem corretamente):

```
digitalWrite(LoadPin, HIGH);  // trava os dados
digitalWrite(ClockPin, LOW);
```

Na sequência, temos a função `scroll()`, que exibe no display os caracteres apropriados da string de texto. A função aceita dois parâmetros: o primeiro é a string que você deseja exibir, e o segundo, a velocidade na qual você deseja que a rolagem ocorra entre as atualizações, em milissegundos:

```
void scroll(char myString[], int speed) {
```

Depois, duas variáveis de tipo `byte` são preparadas; elas armazenarão uma das oito linhas de padrões de bits que compõem o caractere específico sendo exibido:

```
byte firstChrRow, secondChrRow;
```

Outra variável `byte` é declarada, `ledOutput`, que armazenará o resultado de um cálculo realizado sobre o primeiro e o segundo padrão de bits dos caracteres, e decidirá quais LEDs devem estar acesos ou apagados (isso será explicado em breve):

```
byte ledOutput;
```

Mais uma variável de tipo `byte` é declarada, `chrPointer`, e inicializada como 0. Ela armazenará a posição atual na string de texto que está sendo exibida, iniciando em 0 e incrementando até alcançar o comprimento da string:

```
byte chrPointer = 0; // Inicializa o ponteiro de posição da string
```

Outras duas variáveis `byte` são declaradas, que armazenarão o caractere atual e o seguinte na string:

```
byte Char1, Char2;  // os dois caracteres que serão exibidos
```

Essas variáveis são diferentes de `firstChrRow` e `secondChrRow`, pois armazenam o valor ASCII (American Standard Code for Information Interchange) do caractere atual, a ser exibido, e do seguinte na string, enquanto `firstChrRow` e `secondChrRow` armazenam o padrão de bits que forma as letras a serem exibidas.

Todas as letras, números, símbolos etc., que podem ser exibidos em uma tela de computador ou enviados via linha serial, têm um código ASCII, que é simplesmente um número de índice, indicando um caractere correspondente na tabela ASCII. Os caracteres de 0 a 31 são códigos de controle e não serão utilizados, uma vez que não podem ser exibidos em seu display de matriz de pontos. Você utilizará os caracteres ASCII de 32 a 126, os 95 caracteres que podem ser impressos, iniciando no número 32, que corresponde a um espaço, e indo até o 126, o símbolo de til (~). Os caracteres ASCII imprimíveis estão listados na tabela 7.6.

Tabela 7.6 – Caracteres ASCII imprimíveis

```
!"#$%&'()*+,-./0123456789:;ó?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
```

Outra variável byte é declarada e inicializada como 0. Ela armazenará a quantidade de bits, do padrão de caracteres do conjunto atual de letras, que devem ser deslocados para dar a impressão de que o texto está rolando da direita para a esquerda:

```
byte scrollBit = 0;
```

Mais uma variável `byte` armazenará o comprimento da string de caracteres. Ela é inicializada como 0:

```
byte strLength = 0;
```

Então, duas variáveis de tipo `unsigned long` são declaradas. Uma armazenará o tempo atual, em milissegundos, transcorrido desde que o chip do Arduino for inicializado ou reinicializado, e outra, o mesmo valor, só que dessa vez depois de executar uma rotina `while`. Quando tomadas em conjunto, essas variáveis garantem que os bits sejam deslocados apenas depois de transcorrido um intervalo de tempo especificado em milissegundos, para que a rolagem ocorra em uma velocidade que permita a leitura do texto:

```
unsigned long time;
unsigned long counter;
```

Agora, você tem de descobrir quantos caracteres há na string. Há muitas formas de fazê-lo, mas, em seu caso, você simplesmente prepara um loop `while`, que verifica se há dados no array atual de índice, `strLength` (inicializado como 0), e, se afirmativo,

incrementa a variável `strLength` em uma unidade. O loop então se repete até que a condição de `myString[strLength]` seja falsa, ou seja, quando não houver mais caracteres na string e `strLength`, incrementada em uma unidade a cada iteração, armazenar o comprimento da string:

```
while (myString[strLength]) {strLength++;}
```

Na sequência, você define o valor de `counter` como o valor de `millis()`. Você viu `millis()` no projeto 4, e sabe que ela armazena o valor, em milissegundos, do tempo transcorrido desde que o Arduino foi ligado ou reinicializado:

```
counter = millis();
```

Agora, um loop `while` é executado, com a condição de que a posição atual do caractere seja menor do que o comprimento da string, menos um:

```
while (chrPointer < (strLength-1)) {
```

A variável `time` é definida com o valor atual de `millis()`:

```
time = millis();
```

Então, uma instrução `if` verifica se o tempo atual é maior do que o último tempo armazenado, mais o valor em `speed`, ou seja, 45 milissegundos, e, se afirmativo, executa seu bloco de código:

```
if (time > (counter + speed)) {
```

`Char1` é carregada com o valor do caractere ASCII indicado em `chrPointer` no array `myString`, e `Char2` com o valor seguinte:

```
Char1 = myString[chrPointer];
Char2 = myString[chrPointer+1];
```

Agora, um loop `for` faz a iteração, percorrendo cada uma das oito linhas:

```
for (byte y= 0; y<8; y++) {
```

Em seguida, você lê o array `font` e coloca o padrão de bits da linha atual, de um total de oito, em `firstChrRow`, e a segunda linha em `secondChrRow`. Lembre-se de que o array `font` está armazenando os padrões de bits que formam os caracteres na tabela ASCII, mas apenas os referentes aos caracteres que podem ser impressos, de 32 a 126. O primeiro elemento do array é o código ASCII do caractere (menos 32, uma vez que você não está utilizando os caracteres de 0 a 31), e o segundo elemento armazena as oito linhas de padrões de bits que compõem esse caractere. Por exemplo, as letras `A` e `Z` são os caracteres ASCII 65 e 90, respectivamente. Você subtrai desses números um valor de 32, para obter o índice de seu array.

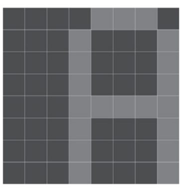
Assim, a letra `A`, código ASCII 65, é armazenada no elemento 33 do array (65 - 32), e a segunda dimensão do array naquele índice armazena os oito padrões de bits que formam essa letra. A letra `Z`, código ASCII 90, é o número de índice 58 no array. Os dados em `font[33][0...8]`, para a letra `A`, correspondem a:

```
{B00001110, B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001},
```

Se você posicionar esses dados individualmente, para analisá-los mais claramente, terá como resultado:

```
B00001110
B00010001
B00010001
B00010001
B00011111
B00010001
B00010001
B00010001
```

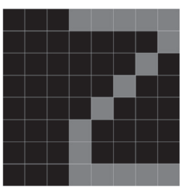
Se analisar mais de perto, você verá o padrão seguinte, que forma a letra `A`:



Para a letra `Z`, os dados no array são:

```
B00011111
B00000001
B00000010
B00000100
B00001000
B00010000
B00010000
B00011111
```

O que corresponde ao seguinte padrão de bits no LED:



Para ler esse padrão de bits, você tem de acessar a fonte de texto, armazenada no espaço de programa, e não na SRAM, como costuma ocorrer. Para tanto, você usa um dos utilitários da biblioteca pgmspace, `pgm_read_byte`:

```
firstChrRow = pgm_read_byte(&font[Char1 - 32][y]);
secondChrRow = (pgm_read_byte(&font[Char2 - 32][y])) << 1;
```

Quando você acessa o espaço de programa, obtém dados armazenados na memória flash. Para fazê-lo, você deve saber o endereço na memória em que os dados estão armazenados (cada local de armazenamento na memória tem um número de endereço único).

Para isso, você utiliza o símbolo `&` à frente de uma variável. Quando o faz, não lê os dados nessa variável, mas, sim, o endereço em que eles estão armazenados. O comando `pgm_read_byte` tem de saber o endereço na memória flash dos dados que você deseja recuperar, por isso você coloca um símbolo `&` à frente de `font[Char1 - 32][y]`, criando `pgm_read_byte(&font[Char1 - 32][y])`. Isso significa, simplesmente, que você lê o byte no espaço de programa armazenado no endereço de `font[Char1 - 32][y]`.

O valor de `secondChrRow` é deslocado uma posição para a esquerda, simplesmente para fazer com que o intervalo entre as letras seja menor, dessa forma facilitando sua leitura no display. Isso ocorre porque, em todos os caracteres, há bits não utilizados à esquerda por três espaços. Você poderia utilizar um deslocamento de duas posições para aproximá-las ainda mais, mas isso dificultaria a leitura.

A linha seguinte carrega o padrão de bits, da linha relevante, em `ledOutput`:

```
ledOutput = (firstChrRow << scrollBit) | (secondChrRow >> (8 - scrollBit) );
```

Como você deseja que as letras tenham rolagem horizontal da direita para a esquerda, você desloca para a esquerda a primeira letra, pela quantidade de vezes em `scrollBit`, e a segunda letra para a direita, por `8 - scrollBit` vezes. Então você aplica um OU lógico aos resultados, para mesclá-los no padrão de 8 bits necessário para exibição. Por exemplo, se as letras exibidas fossem `A` e `Z`, então os padrões para ambas seriam:

```
B00001110 B00011111
B00010001 B00000001
B00010001 B00000010
B00010001 B00000100
B00011111 B00001000
B00010001 B00010000
B00010001 B00010000
B00010001 B00011111
```

Assim, o cálculo anterior na primeira linha, quando `scrollBit` está definida como `5` (ou seja, quando as letras rolaram cinco pixels para a esquerda), seria:

```
B11000000 B00000011
```

O que corresponde à primeira linha de `A` deslocada para a esquerda cinco vezes, e à primeira linha do `Z` deslocada para a direita três vezes (8 - 5). Você pode ver que o padrão da esquerda é o que você obtém ao deslocar a letra `A` cinco pixels para a esquerda, e o padrão da direita é o que você obteria se a letra `Z` rolasse para a direita por três pixels. O OU lógico, símbolo `|`, tem o efeito de mesclar esses dois padrões, criando:

```
B11000011
```

que corresponde ao resultado obtido se as letras `A` e `Z` estivessem lado a lado, e rolassem cinco pixels para a esquerda.

A linha seguinte carrega esse padrão de bits na linha apropriada do buffer de tela:

```
buffer[y] = ledOutput;
```

`scrollBit` é acrescida em uma unidade:

```
scrollBit++;
```

Então uma instrução `if` verifica se o valor de `scrollBit` atingiu `7`. Se afirmativo, ela o define novamente como 0 e eleva `chrPointer` em uma unidade, para que, da próxima vez que for chamada, a função exiba os dois próximos conjuntos de caracteres:

```
if (scrollBit > 6) {
  scrollBit = 0;
  chrPointer++;
}
```

Por fim, o valor de `counter` é atualizado com o valor mais recente de `millis()`:

```
counter = millis();
```

A função `screenUpdate()` simplesmente toma as oito linhas de padrões de bits que você carregou nos oito elementos do array de buffer, e escreve esses dados no chip; o chip, por sua vez, exibe o resultado na matriz:

```
void screenUpdate() {
  for (byte row = 0; row < 8; row++) {
    writeData(row+1, buffer[row]);
  }
}
```

Depois de preparar essas seis funções, você finalmente atinge as funções `setup()` e `loop()` do programa. Na primeira, o chip é inicializado chamando `initMAX7219()`, um timer é criado e definido com um período de atualização de 10 mil microssegundos, e a função `screenUpdate()` é anexada. Assim como antes, isso garante que `screenUpdate()` seja ativada a cada 10 mil microssegundos, independentemente do que mais estiver ocorrendo.

```
void setup() {
  initMAX7219();
  Timer1.initialize(10000);  // inicializa timer1 e define o período de interrupção
  Timer1.attachInterrupt(screenUpdate);
}
```

Por fim, o loop principal do programa tem apenas quatro linhas. A primeira limpa o display, e as três seguintes chamam a rotina de rolagem, para exibir as três linhas de texto e fazê-las rolar horizontalmente no display.

```
void loop() {
  clearDisplay();
  scroll("   BEGINNING ARDUINO   ", 45);
  scroll("   Chapter 7 - LED Displays   ", 45);
  scroll("   HELLO WORLD!!!  :)   ", 45);
}
```

É evidente que você pode alterar o texto no código, fazendo com que o display apresente a mensagem que você quiser. O projeto 21 foi bem complexo no que se refere ao seu código. Como eu disse no início, todo esse trabalho poderia ter sido evitado se você tivesse, simplesmente, utilizado uma das bibliotecas preexistentes para matriz de LEDs, disponíveis em domínio público. Mas, ao fazê-lo, você não teria aprendido como funciona o chip MAX7219, nem como controlá-lo. Essas habilidades podem ser empregadas em praticamente qualquer outro CI externo, uma vez que os princípios envolvidos são muito semelhantes.

No próximo projeto, você utilizará essas bibliotecas, e verá como elas facilitam sua vida. Vamos pegar seu display e nos divertir um pouco com ele.

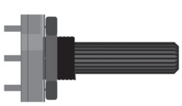
Projeto 22 – Display de matriz de pontos LED – Pong

O projeto 21 não foi fácil, e apresentou a você muitos conceitos novos. Assim, para o projeto 22, você criará um simples joguinho, utilizando o display de matriz de pontos e um potenciômetro. Dessa vez, você utilizará uma das muitas bibliotecas disponíveis para controle de displays de matriz de pontos LED, e verá como elas facilitam seu trabalho de codificação.

Componentes necessários

Os mesmo do projeto 21, mais:

Potenciômetro de 10 kΩ



Conectando os componentes

Deixe o circuito como no projeto 21 e adicione um potenciômetro. Os pinos da esquerda e da direita vão para o terra e para os +5 V, respectivamente, e o pino central vai para o pino analógico 5.

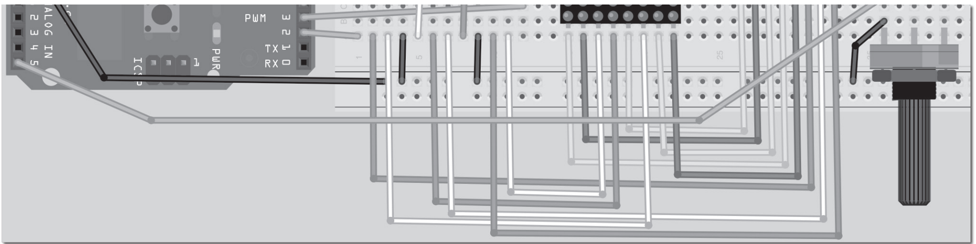


Figura 7.8 – Adicione um potenciômetro ao circuito do projeto 21 (consulte o site da Novatec para versão colorida).

Upload do código

Faça o upload do código da listagem 7.4. Quando o programa for executado, uma bola iniciará em um ponto aleatório na esquerda e avançará para a direita. Utilizando o potenciômetro, você controla a raquete para rebater a bola de volta à parede. Conforme o tempo passa, a velocidade da bola aumenta cada vez mais, até que você não consegue mais acompanhá-la.

Quando a bola ultrapassar a raquete, a tela piscará e o jogo reiniciará. Veja por quanto tempo você consegue acompanhar a bola, até que o jogo reinicie.

Listagem 7.4 – Código para o projeto 22

```
//Projeto 22
#include "LedControl.h"

LedControl myMatrix = LedControl(2, 4, 3, 1);         // cria uma instância de uma Matriz

int column = 1, row = random(8)+1;     // decide em que ponto a bola deve iniciar
int directionX = 1, directionY = 1;     // certifica-se de que ela vai primeiro da esquerda para
                                        // a direita
int paddle1 = 5, paddle1Val;        // pino e valor do potenciômetro
int speed = 300;
int counter = 0, mult = 10;

void setup() {
  myMatrix.shutdown(0, false);   // habilita o display
  myMatrix.setIntensity(0, 8);   // define o brilho como médio
  myMatrix.clearDisplay(0);      // limpa o display
  randomSeed(analogRead(0));
}
```

```
void loop() {
  paddle1Val = analogRead(paddle1);
  paddle1Val = map(paddle1Val, 200, 1024, 1,6);
  column += directionX;
  row += directionY;
  if (column == 6 && directionX == 1 && (paddle1Val == row || paddle1Val+1 == row ||
    paddle1Val+2 == row)) {directionX = -1;}
  if (column == 0 && directionX == -1 ) {directionX = 1;}
  if (row == 7 && directionY == 1 ) {directionY = -1;}
  if (row == 0 && directionY == -1 ) {directionY = 1;}
  if (column == 7) { oops(); }
  myMatrix.clearDisplay(0);  // limpa a tela para o próximo quadro de animação
  myMatrix.setLed(0, column, row, HIGH);
  myMatrix.setLed(0, 7, paddle1Val, HIGH);
  myMatrix.setLed(0, 7, paddle1Val+1, HIGH);
  myMatrix.setLed(0, 7, paddle1Val+2, HIGH);
  if (!(counter % mult)) {speed -= 5; mult * mult;}
  delay(speed);
  counter++;
}

void oops() {
  for (int x=0; x<3; x++) {
  myMatrix.clearDisplay(0);
  delay(250);
    for (int y=0; y<8; y++) {
      myMatrix.setRow(0, y, 255);
    }
    delay(250);
  }
  counter=0;        // reinicia todos os valores
  speed=300;
  column=1;
  row = random(8)+1;   // escolhe uma nova posição inicial
}
```

Projeto 22 – Display de matriz de pontos LED – Pong – Análise do código

O código para o projeto 22 é muito simples. Afinal, estamos nos recuperando do trabalho que tivemos no projeto 21.

Primeiro, inclua a biblioteca `LedControl.h` em seu sketch. Como antes, você terá de baixá-la e instalá-la na pasta `libraries`. A biblioteca, assim como informações adicionais, pode ser encontrada em *www.arduino.cc/playground/Main/LedControl*.

```
#include "LedControl.h"
```

Depois, crie uma instância de um objeto `LedControl`, da seguinte maneira:

```
LedControl myMatrix = LedControl(2, 4, 3, 1);        // cria uma instância de uma Matriz
```

Isso cria um objeto `LedControl`, `myMatrix`. O objeto `LedControl` requer quatro parâmetros. Os primeiros três são números de pinos para o MAX7219, na ordem Data In, Clock e Load. O número final é o número do chip (caso você esteja controlando mais de um MAX7219 e mais de um display).

Depois, você decide em qual coluna e linha a bola iniciará. A linha é escolhida utilizando um número aleatório.

```
int column = 1, row = random(8)+1;  // decide em que ponto a bola deve iniciar
```

Agora, dois inteiros são declarados, para decidir a direção em que a bola viajará. Se o número for positivo, ela avançará da esquerda para a direita e da base para o topo; se negativo, ela fará o caminho inverso.

```
int directionX = 1, directionY = 1;    // certifica-se de que ela vai primeiro da esquerda para
                                       // a direita
```

Você, então, decide qual pino está sendo utilizado para a raquete (o potenciômetro), e declara um inteiro para armazenar o valor lido no pino analógico:

```
int paddle1 = 5, paddle1Val;    // pino e valor do potenciômetro
```

A velocidade da bola é declarada em milissegundos:

```
int speed = 300;
```

Depois você declara e inicializa um contador como `0`, e seu multiplicador como `10`:

```
int counter = 0, mult = 10;
```

A função `setup()` habilita o display, assegurando que o modo de economia de energia esteja definido como `false`. A intensidade é definida como média e limpamos o display, deixando-o preparado para receber o jogo. Antes de você iniciar, `randomSeed` é definida com um valor aleatório, lido a partir de um pino analógico não utilizado.

```
void setup() {
  myMatrix.shutdown(0, false);   // habilite o display
  myMatrix.setIntensity(0, 8);   // Defina o brilho como médio
  myMatrix.clearDisplay(0);      // limpe o display
  randomSeed(analogRead(0));
}
```

No loop principal, você inicia lendo o valor analógico da raquete:

```
paddle1Val = analogRead(paddle1);
```

Então, esses valores são mapeados entre `1` e `6`:

```
paddle1Val = map(paddle1Val, 200, 1024, 1,6);
```

O comando `map` requer cinco parâmetros. O primeiro é o número a ser mapeado. Depois, temos os valores mínimo e máximo do número, e os valores menor e maior aos quais você deseja mapear. Em seu caso, você está tomando o valor em `paddle1Val`, que é a voltagem lida no pino analógico 5. Esse valor vai de `0`, em 0 V, a `1024` em 5 V. Você deseja que esses números sejam mapeados para o intervalo de 1 a 6, uma vez que essas são as linhas em que a raquete será exibida quando desenhada no display.

As coordenadas de coluna e linha são aumentadas pelos valores em `directionX` e `directionY`:

```
column += directionX;
row += directionY;
```

Agora, você tem de decidir se a bola atingiu uma parede ou a raquete e, se afirmativo, deve rebatê-la (a exceção sendo quando ela ultrapassa a raquete). A primeira instrução `if` verifica se a bola atingiu a raquete. Isso é feito decidindo se a coluna da bola está na coluna 6 e (E lógico, `&&`) se ela também está avançando da esquerda para a direita:

```
if (column == 6 && directionX == 1 && (paddle1Val == row || paddle1Val+1 == row
  || paddle1Val+2 == row)) {directionX = -1;}
```

Há três condições que devem ser atendidas para que a direção da bola se altere. A primeira é a de que a coluna seja 6, a segunda, de que a direção seja positiva (ou seja, da esquerda para a direita), e a terceira, de que a bola esteja na mesma linha de qualquer um dos três pontos que formam a raquete. Isso é feito aninhando um conjunto de comandos OU lógico (||) dentro de parênteses. O resultado desse cálculo é verificado primeiro, e depois adicionado às três operações `&&` no primeiro conjunto de parênteses.

Os três conjuntos de instruções `if`, a seguir, verificam se a bola atingiu as paredes do topo, da base ou do lado esquerdo, e, se afirmativo, invertem a direção da bola:

```
if (column == 0 && directionX == -1 ) {directionX = 1;}
if (row == 7 && directionY == 1 ) {directionY = -1;}
if (row == 0 && directionY == -1 ) {directionY = 1;}
```

Por fim, se a bola está na coluna 7, obviamente não atingiu a raquete, mas a ultrapassou. Se esse for o caso, chame a função `oops()` para piscar o display e reinicializar os valores:

```
if (column == 7) { oops(); }
```

Em seguida, limpamos o display para apagar pontos prévios:

```
myMatrix.clearDisplay(0);  // limpa a tela para o próximo frame de animação
```

A bola é desenhada na localização da coluna e da linha. Isso é feito com o comando `.setLed` da biblioteca LedControl:

```
myMatrix.setLed(0, column, row, HIGH);
```

O comando `.setLed` requer quatro parâmetros. O primeiro é o endereço do display. Depois, temos as coordenadas x e y (ou de coluna e linha) e, finalmente, um `HIGH` ou `LOW` para estados de ligado e desligado. Esses parâmetros são utilizados para desenhar os três pontos que formam a raquete na coluna 7 e na linha de valor `paddle1Val` (somando um além dela, e mais um depois disso).

```
myMatrix.setLed(0, 7, paddle1Val, HIGH);
myMatrix.setLed(0, 7, paddle1Val+1, HIGH);
myMatrix.setLed(0, 7, paddle1Val+2, HIGH);
```

Então, você verifica se o módulo de `counter % mult` não (NÃO lógico, `!`) é verdadeiro e, caso isso ocorra, diminui a velocidade em cinco, e multiplica o multiplicador por ele mesmo. Módulo é o resto obtido quando você divide um inteiro por outro. Em seu caso, você divide `counter` por `mult` e verifica se o resto é um número inteiro ou não. Isso basicamente garante que a velocidade apenas aumente depois de um intervalo de tempo definido, e que o tempo se eleve em proporção ao delay decrescente.

```
if (!(counter % mult)) {speed -= 5; mult * mult;}
```

Uma espera em milissegundos com o valor de `speed` é ativada, e o valor de `counter`, acrescido em uma unidade:

```
delay(speed);
counter++;
}
```

Por fim, a função `oops()` provoca um loop `for` dentro de outro loop `for` para limpar o display, e depois preenche todas as linhas repetidamente, com um delay de 250 milissegundos entre cada operação. Isso faz com que todos os LEDs pisquem, acendendo e apagando para indicar que a bola saiu de jogo e que o jogo está prestes a reiniciar. Então, todos os valores de `counter`, `speed` e `column` são definidos novamente em suas posições de início, e um novo valor aleatório é escolhido para `row`.

```
void oops() {
  for (int x=0; x<3; x++) {
  myMatrix.clearDisplay(0);
  delay(250);
    for (int y=0; y<8; y++) {
      myMatrix.setRow(0, y, 255);
    }
    delay(250);
  }
  counter=0;        // reinicia todos os valores
  speed=300;
  column=1;
  row = random(8)+1;   // escolhe uma nova posição de início
}
```

O comando `.setRow` opera transmitindo o endereço do display, o valor da linha e o padrão binário que indica quais dos LEDs devem acender e apagar. Nesse caso, você deseja todos acesos, o que, em binário, é representado por 11111111, e em decimal, por 255.

O propósito do projeto 22 foi mostrar como é mais fácil controlar um chip controlador de LEDs se você empregar uma biblioteca de código projetada para o chip. No projeto 21 você escolheu o caminho mais difícil, codificando todos os elementos a partir do zero; no projeto 22, o trabalho pesado foi todo feito nos bastidores. Há outras bibliotecas de matriz disponíveis; de fato, o próprio IDE do Arduino vem com uma, a `matrix`. Como tive melhores resultados com a biblioteca `LedControl.h`, preferi utilizá-la. Utilize a biblioteca que melhor atender às suas necessidades.

No próximo capítulo, você verá um tipo diferente de matriz de pontos, o LCD.

EXERCÍCIO

Tome os conceitos dos projetos 21 e 22 e combine-os. Crie um jogo Pong, mas faça com que o código mantenha uma pontuação (determinada pelos milissegundos transcorridos desde que a partida iniciou). Quando a bola sair de jogo, utilize a função de texto com rolagem horizontal para mostrar a pontuação da partida que acabou de terminar (os milissegundos que o jogador sobreviveu), e a pontuação mais alta até então.

Resumo

O capítulo 7 apresentou alguns tópicos bem complexos, incluindo o uso de CIs externos. Você ainda não completou nem a metade dos projetos, e já sabe como controlar um display de matriz de pontos utilizando tanto registradores de deslocamento quanto um CI controlador de LED dedicado. Assim, também, você primeiro aprendeu a codificar suas criações da forma mais difícil, para, em seguida, criar código com facilidade, incorporando uma biblioteca projetada especificamente para seu CI controlador de LED.

Você também aprendeu o conceito, por vezes desconcertante, da multiplexação, uma habilidade que será de grande utilidade em muitas outras situações, além do uso em displays de matriz de pontos.

Assuntos e conceitos abordados no capítulo 7:

- como conectar um display de matriz de pontos;
- como instalar uma biblioteca externa;
- o conceito da multiplexação (multiplexing, ou muxing);
- como utilizar a multiplexação para acender 64 LEDs individualmente, utilizando apenas 16 pinos de saída;

- o conceito básico de timers;
- como utilizar a biblioteca TimerOne para ativar códigos específicos, independentemente do que mais estiver acontecendo;
- como incluir bibliotecas externas em seu código utilizando a diretiva `#include`;
- como utilizar números binários para armazenar imagens LED;
- como inverter um número binário utilizando um NÃO bit a bit (~);
- como aproveitar a persistência da visão para enganar os olhos;
- como armazenar quadros de animação em arrays multidimensionais;
- como declarar e inicializar um array multidimensional;
- como acessar dados em um elemento específico de um array bidimensional;
- como realizar uma rotação bit a bit (ou deslocamento circular);
- como controlar displays de matriz de pontos LED utilizando registradores de deslocamento;
- como controlar displays de matriz de pontos LED utilizando CIs MAX7219;
- como coordenar corretamente os pulsos para entrada e saída de dados em CIs externos;
- como armazenar uma fonte de caracteres em um array bidimensional;
- como utilizar os registros do MAX7219;
- como ir além dos limites da SRAM e armazenar dados no espaço de programa;
- como inverter a ordem dos bits;
- como fazer a rolagem de textos e de outros símbolos em um display de matriz de pontos;
- o conceito da tabela de caracteres ASCII;
- como escolher um caractere a partir de seu código ASCII;
- como descobrir o tamanho de uma string de texto;
- como escrever e ler dados no espaço de programa;
- como obter o endereço na memória de uma variável utilizando o símbolo `&`;
- como utilizar a biblioteca `LedControl.h` para controlar LEDs individuais e linhas de LEDs;
- como utilizar operadores lógicos;
- como facilitar sua vida, e desenvolver código mais rapidamente, utilizando bibliotecas de código.

CAPÍTULO 8

Displays de cristal líquido

Vamos investigar outro método muito comum de exibição de textos e símbolos, os LCDs (Liquid Crystal Display, ou display de cristal líquido). LCDs são displays tipicamente utilizados em calculadoras e despertadores. Muitos projetos do Arduino envolvem LCDs, por isso é essencial que você saiba como utilizá-los. Displays de LCD requerem chips controladores para controlá-los; esses chips são integrados ao display. O tipo de chip controlador mais popular é o Hitachi HD44780 (ou compatível).

A criação de projetos com base em displays LCD é muito simples, graças a um conjunto prontamente disponível de bibliotecas de código para LCDs. O IDE do Arduino vem com uma biblioteca desse tipo, a `LiquidCrystal.h`, que tem uma grande lista de recursos. Essa é a biblioteca que você utilizará em seus projetos.

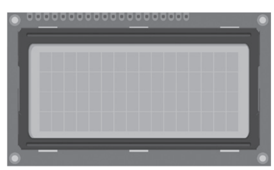
Projeto 23 – Controle básico de um LCD

Para iniciar, você criará um projeto de demonstração, exibindo a maioria das funções disponíveis na biblioteca `LiquidCrystal.h`. Para tanto, utilizaremos um display LCD 16 x 2 com backlight.

Componentes necessários

Você deve adquirir um display LCD que utilize o controlador HD44780. Há muitos disponíveis, e em todos os tipos de cores. Como astrônomo amador, prefiro os displays de vermelho sobre preto (texto vermelho sobre fundo preto), uma vez que estes preservam a visibilidade à noite, caso venham a ser utilizados em projetos de astronomia. Você pode escolher outra cor para o texto ou para o fundo, mas seu display deve ter uma backlight e ser capaz de exibir 16 colunas e duas linhas de caracteres (muitas vezes referidos como displays LCD 16 x 2).

LCD 16 x 2 com backlight



Resistor limitador de corrente (backlight)



Resistor limitador de corrente (contraste) 

Conectando os componentes

O circuito para o projeto 23 é muito simples. Encontre o datasheet para o LCD que você está utilizando. Os pinos do Arduino, dos +5 V e do terra, que mostramos a seguir (Tabela 8.1), devem ser conectados ao LCD.

Tabela 8.1 – Pinos utilizados para o LCD

Arduino	Outro	Matriz
Digital 11		Enable (ativação)
Digital 12		RS (seleção de registrador)
Digital 5		DB4 (pino de dados 4)
Digital 4		DB5 (pino de dados 5)
Digital 3		DB6 (pino de dados 6)
Digital 2		DB7 (pino de dados 7)
Terra (Gnd)		Vss (GND)
Terra (Gnd)		R/W (leitura/escrita)
+5 V		Vdd
	+5 V via resistor	Vo (contraste)
	+5 V via resistor	A/Vee (alimentação para o LED)
	Terra (Gnd)	Terra (Gnd) para o LED

Os pinos de dados (Data Pins) de 0 a 3 não são usados, pois você utilizará aquilo que é conhecido como modo de 4 bits. Para um display LCD típico, o circuito da figura 8.1 estará correto.

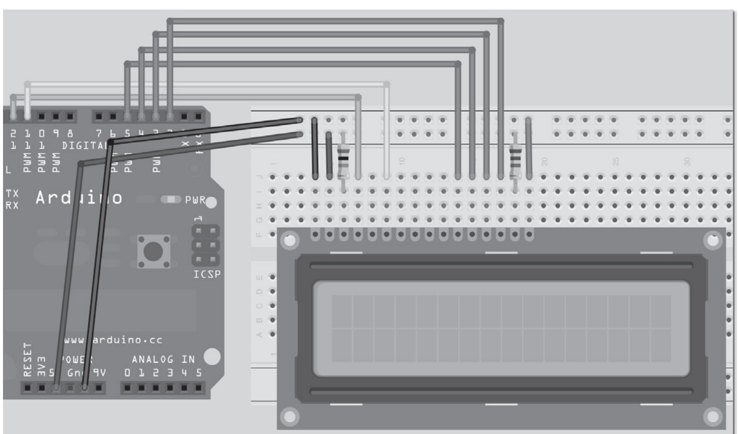


Figura 8.1 – Circuito para o Projeto 23 – Controle básico de um LCD (consulte o site da Novatec para versão colorida).

O pino do ajuste de contraste do LCD deve ser conectado via um resistor limitador de corrente, para ajustar o contraste em um nível desejado. Um valor de cerca de 10 kΩ deve ser suficiente. Se você tiver dificuldades em encontrar o valor correto, conecte um potenciômetro (com valor entre 4 kΩ e 10 kΩ) com o terminal esquerdo indo para os +5 V, o direito para o terra, e o terminal central para o pino de ajuste de contraste (pino 3 no LCD de teste). Agora, você pode utilizar o botão do potenciômetro para ajustar o contraste até que possa ver o display com clareza.

O backlight em meu LCD de teste necessitou de 4,2 V, por isso adicionei o resistor limitador de corrente apropriado entre os +5 V e o pino de alimentação do LED (pino 15 em meu LCD). Poderíamos ter conectado o pino de alimentação do LED a um pino PWM do Arduino e utilizado uma saída PWM para controlar o brilho da backlight, mas, por simplicidade, não utilizaremos esse método neste projeto. Assim que você estiver satisfeito com a conexão dos pinos do Arduino, dos +5 V e do terra (de acordo com o datasheet do LCD), você pode digitar o código.

Digite o código

Verifique sua fiação, e depois, faça o upload do código da listagem 8.1.

Listagem 8.1 – Código para o projeto 23

```
// PROJETO 23
#include <LiquidCrystal.h>

// Inicializa a biblioteca com os números dos pinos de interface
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Cria um objeto lcd e atribui os pinos

void setup() {
  lcd.begin(16, 2);   // Define o display com 16 colunas e 2 linhas
}

void loop() {
  // Executa as sete rotinas de demonstração
  basicPrintDemo();
  displayOnOffDemo();
  setCursorDemo();
  scrollLeftDemo();
  scrollRightDemo();
  cursorDemo();
  createGlyphDemo();
}

void basicPrintDemo() {
  lcd.clear();          // Limpa o display
```

```
  lcd.print("Basic Print");  // Imprime algum texto
  delay(2000);
}

void displayOnOffDemo() {
  lcd.clear();            // Limpa o display
  lcd.print("Display On/Off");   // Imprime algum texto
  for(int x=0; x < 3; x++) {     // Itera 3 vezes
    lcd.noDisplay();      // Apaga o display
    delay(1000);
    lcd.display();          // Acende-o novamente
    delay(1000);
  }
}

void setCursorDemo() {
  lcd.clear();          // Limpa o display
  lcd.print("SetCursor Demo"); // Imprime algum texto
  delay(1000);
  lcd.clear();          // Limpa o display
  lcd.setCursor(5,0);        // Cursor na coluna 5, linha 0
  lcd.print("5,0");
  delay(2000);
  lcd.setCursor(10,1);     // Cursor na coluna 10, linha 1
  lcd.print("10,1");
  delay(2000);
  lcd.setCursor(3,1);        // Cursor na coluna 3, linha 1
  lcd.print("3,1");
  delay(2000);
}

void scrollLeftDemo() {
  lcd.clear();          // Limpa o display
  lcd.print("Scroll Left Demo");
  delay(1000);
  lcd.clear();          // Limpa o display
  lcd.setCursor(7,0);
  lcd.print("Beginning");
  lcd.setCursor(9,1);
  lcd.print("Arduino");
  delay(1000);
  for(int x=0; x<16; x++) {
    lcd.scrollDisplayLeft();   // Rola o display 16 vezes para a esquerda
    delay(250);
  }
}
```

```
void scrollRightDemo() {
  lcd.clear();          // Limpa o display
  lcd.print("Scroll Right");
  lcd.setCursor(0,1);
  lcd.print("Demo");
  delay(1000);
  lcd.clear();          // Limpa o display
  lcd.print("Beginning");
  lcd.setCursor(0,1);
  lcd.print("Arduino");
  delay(1000);
  for(int x=0; x<16; x++) {
    lcd.scrollDisplayRight();  // Rola o display 16 vezes para a direita
    delay(250);
  }
}

void cursorDemo() {
  lcd.clear();    // Limpa o display
  lcd.cursor();   // Cursor visível
  lcd.print("Cursor On");
  delay(3000);
  lcd.clear();      // Limpa o display
  lcd.noCursor();   // Cursor invisível
  lcd.print("Cursor Off");
  delay(3000);
  lcd.clear();    // Limpa o display
  lcd.cursor();   // Cursor visível
  lcd.blink();    // Cursor piscando
  lcd.print("Cursor Blink On");
  delay(3000);
  lcd.noCursor();   // Cursor invisível
  lcd.noBlink();    // Efeito blink desligado
}


void createGlyphDemo() {
  lcd.clear();

  byte happy[8] = {   // Cria um array de bytes com uma cara feliz
  B00000,
  B00000,
  B10001,
  B00000,
  B10001,
```

```
    B01110,
    B00000,
    B00000};

    byte sad[8] = {   // Cria um array de bytes com uma cara triste
    B00000,
    B00000,
    B10001,
    B00000,
    B01110,
    B10001,
    B00000,
    B00000};

    lcd.createChar(0, happy);  // Cria o caractere personalizado 0
    lcd.createChar(1, sad);    // Cria o caractere personalizado 1

    for(int x=0; x<5; x++) { z   // Executa a animação 5 vezes
      lcd.setCursor(8,0);
      lcd.write(0);              // Escreve o caractere personalizado 0
      delay(1000);
      lcd.setCursor(8,0);
      lcd.write(1);              // Escreve o caractere personalizado 1
      delay(1000);
    }
  }
```

Projeto 23 – Controle básico de um LCD – Análise do código

Primeiramente, você carrega a biblioteca que vai utilizar para controlar o LCD. Há muitas bibliotecas e exemplos de código disponíveis para tipos diferentes de LCDs, e você pode encontrar todas elas no Arduino Playground, em *www.arduino.cc/playground/Code/LCD*. Entretanto, o IDE do Arduino vem com uma biblioteca, a `LiquidCrystal.h`, que é muito fácil de se compreender e utilizar:

```
#include <LiquidCrystal.h>
```

Agora, você deve criar um objeto LiquidCrystal e definir os pinos apropriados:

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);   // Cria um objeto lcd e atribui os pinos
```

Com isso, você criou um objeto LiquidCrystal, dando a ele o nome de `lcd`. Os dois primeiros parâmetros definem os pinos para o RS (register select, ou seleção de registrador) e Enable (ativação). Os quatro últimos parâmetros são os pinos de dados, de D4 a D7. Como você está utilizando o modo de 4 bits, são usados apenas quatro dos oito pinos de dados no display.

A diferença entre o modo de 8 bits e o modo de 4 bits é que, no primeiro, você pode enviar dados um byte de cada vez, enquanto, no segundo, os 8 bits têm de ser divididos em números de 4 bits (conhecidos como *nibbles*). Isso torna o código maior e mais complexo. Entretanto, você está utilizando uma biblioteca pronta, portanto, não tem de se preocupar com isso. Se, por outro lado, você estivesse escrevendo um código no qual tamanho ou desempenho fossem fatores críticos, teria de considerar a possibilidade de escrever diretamente para o LCD em modo de 8 bits. O uso do modo de 4 bits tem a vantagem de economizar quatro pinos, que podem ser úteis, caso você queira conectar outros dispositivos ao mesmo tempo.

No loop `setup()` você inicializa o display com o tamanho necessário, de 16 colunas e 2 linhas:

```
lcd.begin(16, 2); // Define o display com 16 colunas e 2 linhas
```

O loop principal do programa simplesmente executa sete rotinas de demonstração, uma de cada vez, antes de iniciar uma nova iteração. Cada rotina de demonstração utiliza um conjunto de rotinas relacionadas na biblioteca `LiquidCrystal.h`:

```
void loop() {
  // Executa as sete rotinas de demonstração
  basicPrintDemo();
  displayOnOffDemo();
  setCursorDemo();
  scrollLeftDemo();
  scrollRightDemo();
  cursorDemo();
  createGlyphDemo();
}
```

A primeira função é `basicPrintDemo()`, projetada para mostrar o uso do comando `.print()`. Essa demonstração simplesmente limpa o display, utilizando `lcd.clear()`, e depois imprime no display, utilizando `lcd.print()`. Note que, se você tivesse inicializado seu objeto LiquidCrystal com outro nome, por exemplo `LCD1602`, esses comandos seriam `LCD1602.clear()` e `LCD1602.print()`, de acordo. Em outras palavras, o comando segue o nome do objeto, com um ponto entre eles.

O comando `print()` imprime o que estiver dentro dos parênteses, na localização atual do cursor. A localização padrão do cursor é sempre coluna 0, linha 0 (o canto superior esquerdo). Depois de limpar o display, o cursor será definido em sua posição padrão, ou inicial:

```
void basicPrintDemo() {
  lcd.clear();          // Limpa o display
  lcd.print("Basic Print");  // Imprime algum texto
  delay(2000);
}
```

A segunda função é projetada para exibir os comandos `display()` e `noDisplay()`. Esses são comandos que simplesmente ativam ou desativam o display. A rotina imprime `"Display On/Off"` e, então, executa um loop três vezes para desligar o display, esperar por um segundo, ligar o display, esperar por mais um segundo e repetir o processo. Sempre que você desligar o display, o que estiver impresso em sua tela antes de ele ser desligado será preservado quando ele for reativado.

```
void displayOnOffDemo() {
  lcd.clear();          // Limpa o display
  lcd.print("Display On/Off"); // Imprime algum texto
  for(int x=0; x < 3; x++) {   // Itera 3 vezes
    lcd.noDisplay();    // Apaga o display
    delay(1000);
    lcd.display();        // Acende-o novamente
    delay(1000);
  }
}
```

A próxima função exibe o comando `setCursor()`, que posiciona o cursor na coluna e na linha colocadas entre os parênteses. A demonstração posiciona o cursor em três localizações diferentes, imprimindo a localização no display. O comando `setCursor()` é útil para controlar o layout de seu texto e para garantir que sua saída vá para a seção apropriada da tela do display.

```
void setCursorDemo() {
  lcd.clear();          // Limpa o display
  lcd.print("SetCursor Demo"); // Imprime algum texto
  delay(1000);
  lcd.clear();          // Limpa o display
  lcd.setCursor(5,0);       // Cursor na coluna 5, linha 0
  lcd.print("5,0");
  delay(2000);
  lcd.setCursor(10,1);     // Cursor na coluna 10, linha 1
  lcd.print("10,1");
  delay(2000);
  lcd.setCursor(3,1);       // Cursor na coluna 3, linha 1
  lcd.print("3,1");
  delay(2000);
}
```

Há dois comandos na biblioteca para a rolagem de texto: `scrollDisplayLeft()` e `scrollDisplayRight()`. Duas rotinas de demonstração exibem o uso desses comandos. A primeira imprime `"Beginning Arduino"` no lado direito do display e rola o texto 16 vezes para a esquerda, o que fará com que ele saia da tela:

```
void scrollLeftDemo() {
  lcd.clear();    // Limpa o display
```

```
    lcd.print("Scroll Left Demo");
    delay(1000);
    lcd.clear();    // Limpa o display
    lcd.setCursor(7,0);
    lcd.print("Beginning");
    lcd.setCursor(9,1);
    lcd.print("Arduino");
    delay(1000);
    for(int x=0; x<16; x++) {
      lcd.scrollDisplayLeft(); // Rola o display 16 vezes para a esquerda
      delay(250);
    }
  }
```

A função seguinte atua de modo semelhante, iniciando com o texto na esquerda e rolando o texto 16 vezes para a direita, até que ele saia da tela:

```
  void scrollRightDemo() {
    lcd.clear();          // Limpa o display
    lcd.print("Scroll Right");
    lcd.setCursor(0,1);
    lcd.print("Demo");
    delay(1000);
    lcd.clear();          // Limpa o display
    lcd.print("Beginning");
    lcd.setCursor(0,1);
    lcd.print("Arduino");
    delay(1000);
    for(int x=0; x<16; x++) {
      lcd.scrollDisplayRight();  // Rola o display 16 vezes para a direita
      delay(250);
    }
  }
```

Até aqui o cursor esteve invisível — ele sempre esteve lá, mas não podia ser visto. Quando você limpa o display, o cursor retorna para o canto superior esquerdo (coluna 0, linha 0). Depois de imprimir algum texto, o cursor estará posicionado logo depois do último caractere impresso. A função seguinte limpa o display e, então, ativa o cursor com `cursor()` e imprime algum texto. O cursor estará visível logo após esse texto, como um símbolo de underscore (_):

```
  void cursorDemo() {
    lcd.clear();  // Limpa o display
    lcd.cursor(); // Cursor visível
    lcd.print("Cursor On");
    delay(3000);
```

Na sequência, limpamos o display novamente. Dessa vez o cursor é desativado, retornando ao seu modo padrão, utilizando `noCursor()`. Agora o cursor não poderá mais ser visto:

```
    lcd.clear();    // Limpa o display
    lcd.noCursor();   // Cursor invisível
    lcd.print("Cursor Off");
    delay(3000);
```

Então, ativamos o cursor novamente. O modo blink também é ativado utilizando `blink()`:

```
    lcd.clear();    // Limpa o display
    lcd.cursor();   // Cursor visível
    lcd.blink();    // Cursor piscando
    lcd.print("Cursor Blink On");
    delay(3000);
```

Dessa vez o cursor não apenas estará visível, como também estará piscando, acendendo e apagando. Esse modo é útil, caso você esteja aguardando alguma entrada de texto do usuário. O cursor piscando atuará como uma indicação, solicitando ao usuário que digite algum texto.

Por fim, o cursor e o modo blink são desligados, retornando o cursor para o modo padrão:

```
    lcd.noCursor();   // Cursor invisível
    lcd.noBlink();    // Efeito blink desligado
  }
```

A função final, `createGlyphDemo()`, cria caracteres personalizados. A maioria dos LCDs permite que você programe seus próprios caracteres personalizados. Um LCD normal de 16 x 2 tem espaço para armazenar oito caracteres personalizados na memória. Os caracteres têm 5 pixels de largura e 8 pixels de altura (um pixel é um elemento de imagem [picture element], como os pontos individuais que compõem um display digital). Limpamos o display e, depois, dois arrays de tipo byte são inicializados, com o padrão binário de uma cara feliz e de uma cara triste. Os padrões binários têm 5 bits de largura.

```
  void createGlyphDemo() {
    lcd.clear();

    byte happy[8] = {   // Cria um array de bytes com uma cara feliz
    B00000,
    B00000,
    B10001,
    B00000,
    B10001,
```

```
B01110,
B00000,
B00000};

byte sad[8] = {   // Cria um array de bytes com uma cara triste
B00000,
B00000,
B10001,
B00000,
B01110,
B10001,
B00000,
B00000};
```

Então, você cria os dois caracteres personalizados, utilizando o comando `createChar()`, que requer dois parâmetros; o primeiro é o número do caractere personalizado (de 0 a 7, no caso do meu LCD de teste, que pode armazenar um máximo de oito caracteres), e o segundo parâmetro é o nome do array que será usado como base para criar e armazenar o padrão binário do caractere personalizado na memória do LCD:

```
lcd.createChar(0, happy);  // Cria o caractere personalizado 0
lcd.createChar(1, sad);    // Cria o caractere personalizado 1
```

Agora, um loop `for` será repetido cinco vezes. A cada iteração, o cursor será definido na coluna 8, linha 0, e o primeiro caractere personalizado será escrito nessa localização utilizando o comando `write()`, que escreve o caractere personalizado, declarado dentro dos parênteses, na localização atual do cursor. O primeiro caractere, uma carinha feliz, é escrito na localização do cursor; depois de aguardar um segundo, o segundo caractere, uma carinha triste, é escrito na mesma localização. Repetimos esse processo cinco vezes, para realizar uma animação rudimentar.

```
  for(int x=0; x<5; x++) {   // Executa a animação 5 vezes
    lcd.setCursor(8,0);
    lcd.write(0);        // Escreve o caractere personalizado 0
    delay(1000);
    lcd.setCursor(8,0);
    lcd.write(1);        // Escreve o caractere personalizado 1
    delay(1000);
  }
}
```

O projeto 23 abordou a maioria dos comandos mais populares da biblioteca `LiquidCrystal.h`. Todavia, há muitos outros que podemos utilizar. Você pode ler sobre eles na biblioteca de referência do Arduino, em *www.arduino.cc/en/Reference/LiquidCrystal*.

Projeto 23 – Controle básico de um LCD – Análise do hardware

O novo componente neste projeto é, obviamente, o LCD. Um display de cristal líquido funciona utilizando as propriedades de modulação da luz nos cristais líquidos. O display é formado de pixels, cada um preenchido com cristais líquidos. Esses pixels são dispostos à frente de uma fonte de luz traseira ou de um refletor. Os cristais são posicionados em camadas imprensadas entre filtros polarizadores. Os dois painéis polarizadores são alinhados a 90 graus um do outro, o que bloqueia a luz. O primeiro filtro polarizador polarizará as ondas de luz de modo que elas avancem apenas em uma orientação. O segundo filtro, estando a 90 graus do primeiro, bloqueará a luz. Em outras palavras, imagine que o filtro é composto de fendas muito finas, que correm em uma direção. A luz polarizada percorrerá essas fendas na mesma orientação, mas quando atingir o segundo filtro, com fendas que avançam na outra direção, ela não passará. Aplicando uma corrente às linhas e colunas das camadas, podemos alterar a orientação dos cristais e fazer com que eles se alinhem de acordo com o campo elétrico. Isso faz com que a luz gire (*twist*) 90 graus, permitindo que passe pelo segundo filtro. Por isso, alguns displays são referidos como “Super-Twist”.

O LCD é composto de uma grade de pixels, organizados em grades menores que formam os caracteres. Um LCD típico de 16 x 2 terá 16 grades de caracteres em duas linhas. Cada grade de caracteres é composta de 5 pixels de largura e 8 pixels de altura. Se você aumentar muito o contraste de seu display, os 32 arrays de 5 x 8 pixels se tornarão visíveis.

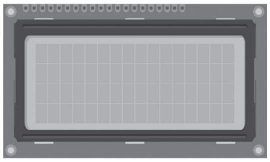
Isso é tudo que você tem de saber sobre o funcionamento dos LCDs. Agora, vamos colocar na prática o que aprendemos, criando um display de temperatura.

Projeto 24 – Display LCD de temperatura

Este projeto é uma demonstração simples do uso de um LCD para apresentar informações úteis ao usuário — nesse caso, a temperatura de um sensor de temperatura analógico. Você adicionará um botão para alternar a exibição entre Celsius e Fahrenheit. Além disso, a temperatura máxima e a mínima serão exibidas na segunda linha.

Componentes necessários

LCD 16 x 2 com backlight



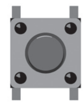
Resistor limitador de corrente (backlight)



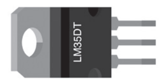
Resistor limitador de corrente (contraste)



Botão



Sensor de temperatura analógico



Conectando os componentes

Utilize exatamente o mesmo circuito definido para o projeto 23. Depois, adicione um botão e um sensor de temperatura, como mostra a figura 8.2.

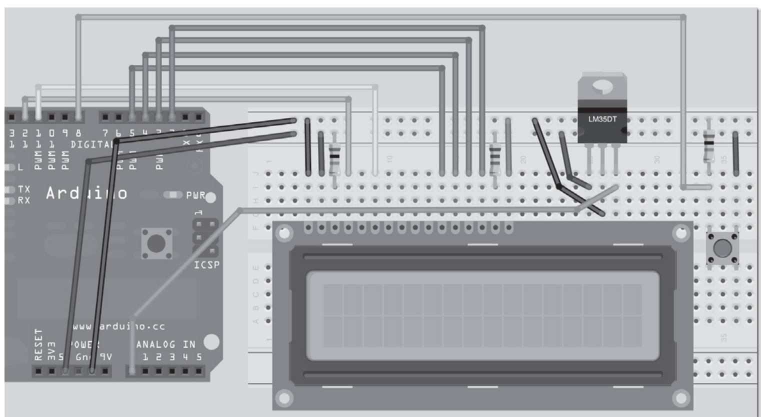


Figura 8.2 – Circuito para o Projeto 24 – Display LCD de temperatura (consulte o site da Novatec para versão colorida).

Utilizei um sensor de temperatura LM35DT, com um alcance de 0°C a 100°C. Você pode utilizar qualquer sensor de temperatura semelhante. O LM35, por exemplo, registra de -55°C a +150°C. Você terá de ajustar seu código de acordo (falarei mais sobre isso posteriormente).

Digite o código

Verifique sua fiação e, depois, faça o upload do código da listagem 8.2.

Listagem 8.2 – Código para o projeto 24

```
// PROJETO 24
#include <LiquidCrystal.h>

// Inicializa a biblioteca com os números dos pinos de interface
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Cria um objeto lcd e atribui os pinos
```

```
int maxC=0, minC=100, maxF=0, minF=212;
int scale = 1;
int buttonPin=8;

void setup() {
  lcd.begin(16, 2);   // Define o display com 16 colunas e 2 linhas
  analogReference(INTERNAL);
  pinMode(buttonPin, INPUT);
  lcd.clear();
}

void loop() {
  lcd.setCursor(0,0);            // Define o cursor na posição de início
  int sensor = analogRead(0);    // Lê a temperatura do sensor
  int buttonState = digitalRead(buttonPin);    // Verifica se o botão foi pressionado
  switch (buttonState) {         // Altera o estado da escala, caso tenha sido pressionado
    case HIGH:
      scale=-scale;              // Inverte a escala
      lcd.clear();
  }

  delay(250);
  switch (scale) {  // Decide por escala em Celsius ou Fahrenheit
    case 1:
      celsius(sensor);
      break;
    case -1:
      fahrenheit(sensor);
  }
}

void celsius(int sensor) {
  lcd.setCursor(0,0);
  int temp = sensor * 0.09765625;   // Converte para Celsius
  lcd.print(temp);
  lcd.write(B11011111);    // Símbolo de grau
  lcd.print("C  ");
  if (temp>maxC) {maxC=temp;}
  if (temp<minC) {minC=temp;}
  lcd.setCursor(0,1);
  lcd.print("H=");
  lcd.print(maxC);
  lcd.write(B11011111);
  lcd.print("C L=");
  lcd.print(minC);
  lcd.write(B11011111);
  lcd.print("C  ");
}
```

```
void fahrenheit(int sensor) {
  lcd.setCursor(0,0);
  float temp = ((sensor * 0.09765625) * 1.8)+32;  // Converte para Fahrenheit
  lcd.print(int(temp));
  lcd.write(B11011111);  // Imprime o símbolo de grau
  lcd.print("F ");
  if (temp>maxF) {maxF=temp;}
  if (temp<minF) {minF=temp;}
  lcd.setCursor(0,1);
  lcd.print("H=");
  lcd.print(maxF);
  lcd.write(B11011111);
  lcd.print("F L=");
  lcd.print(minF);
  lcd.write(B11011111);
  lcd.print("F ");
}
```

Quando você executar o código, a temperatura atual será exibida na linha de cima do LCD. A linha de baixo exibirá a temperatura máxima e a mínima, registradas desde que o Arduino foi ligado, ou desde que o programa foi reinicializado. Pressionando o botão, você pode alterar a escala de temperatura entre Celsius e Fahrenheit.

Projeto 24 – Display LCD de temperatura – Análise do código

Assim como antes, você carrega a biblioteca `LiquidCrystal.h` em seu sketch:

```
#include <LiquidCrystal.h>
```

Inicializamos um objeto LiquidCrystal, `lcd`, e definimos os pinos apropriados:

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);   // Cria um objeto lcd e atribui os pinos
```

Algumas variáveis inteiras, para armazenar as temperaturas máximas e mínimas em graus Celsius e Fahrenheit, são declaradas e inicializadas com valores máximos e mínimos impossíveis. Esses valores serão alterados assim que o programa for executado pela primeira vez.

```
int maxC=0, minC=100, maxF=0, minF=212;
```

Uma variável `scale`, de tipo inteiro, é declarada e inicializada com `1`. A variável `scale` decidirá se você está utilizando Celsius ou Fahrenheit como escala de temperatura. Por padrão, ela é definida como `1`, representando Celsius. Você pode alterar esse valor para `-1`, quando quiser utilizar a escala Fahrenheit.

```
int scale = 1;
```

Uma variável inteira, para armazenar o pino utilizado pelo botão, é declarada e inicializada:

```
int buttonPin=8;
```

No loop `setup()` você define o display com 16 colunas e 2 linhas:

```
lcd.begin(16, 2); // Define o display com 16 colunas e 2 linhas
```

A referência para o pino analógico é definida como `INTERNAL`:

```
analogReference(INTERNAL);
```

Isso resulta em um intervalo maior de valores no ADC (Analog to Digital Converter, conversor analógico para digital) do Arduino. A tensão de saída do LM35DT em 100°C é de 1 V. Se você estivesse utilizando a referência padrão de 5 V, uma temperatura de 50°C, que representa metade do alcance do sensor, acusaria uma leitura de 0,5 V = (0,5 / 5) * 1023 = 102 no ADC, o que representa apenas cerca de 10% de seu intervalo de valores possíveis. Quando utilizamos a voltagem de referência interna de 1,1 V, o valor do pino analógico em 50°C será de 0,5 V = (0,5 / 1,1) * 1023 = 465.

Como você pode ver, isso é praticamente metade do intervalo máximo dos valores que o pino analógico pode ler (de 0 a 1023), portanto, a resolução e a precisão da leitura foram melhoradas, assim como a sensibilidade do circuito.

Na sequência, o pino do botão é definido como entrada, e limpamos o display LCD:

```
pinMode(buttonPin, INPUT);
lcd.clear();
```

No loop principal, o programa inicia definindo o cursor em sua posição inicial:

```
lcd.setCursor(0,0);    // Define o cursor na posição de início
```

Depois, você lê um valor do sensor de temperatura no pino analógico 0:

```
int sensor = analogRead(0);   // Lê a temperatura do sensor
```

Então, você lê o estado do botão e armazena o valor em `buttonState`:

```
int buttonState = digitalRead(buttonPin);   // Verifica se o botão foi pressionado
```

Agora, você tem de descobrir se o botão foi ou não pressionado e, se afirmativo, deve alterar a escala de Celsius para Fahrenheit, ou vice-versa. Isso é feito utilizando uma instrução switch/case:

```
switch (buttonState) {   // Altera o estado da escala, caso tenha sido pressionado
  case HIGH:
    scale=-scale;   // Inverte a escala
    lcd.clear();
}
```

Esse é um novo conceito: o comando `switch/case` controla o fluxo do programa, especificando qual código deve ser executado, com base nas condições que foram atendidas. A instrução `switch/case` compara o valor de uma variável com valores nas instruções `case` e, se verdadeiro, executa o código da instrução correspondente.

Por exemplo, se você tivesse uma variável `var` e quisesse executar determinadas ações, caso seu valor fosse `1`, `2` ou `3`, poderia utilizar o seguinte código:

```
switch (var) {
  case 1:
    // executa este código se var for 1
    break;
  case 2:
    // executa este código se var for 2
    break;
  case 3:
    // executa este código se var for 3
    break;
  default:
    // se nada mais corresponde, execute este código
  }
```

A instrução `switch/case` verificará o valor de `var`. Se ele for `1`, ela executará o código dentro do bloco do caso `1` até o comando `break`. O comando `break` é utilizado para sair da instrução `switch/case`. Sem ele, o código continuaria sendo executado até que um comando `break` fosse atingido ou que se alcançasse o término da instrução `switch/case`. Se nenhum dos valores verificados for atingido, então o código dentro da seção `default` será executado. Note que a seção `default` é opcional e desnecessária.

Aqui, você verifica apenas um caso: se `buttonState` é `HIGH`. Se afirmativo, o valor da escala será invertido (de Celsius para Fahrenheit, ou vice-versa) e limparemos o display.

Na sequência, temos uma pequena espera:

```
delay(250);
```

Seguida de outra instrução `switch/case`, para verificar se o valor de `scale` é `1`, para Celsius, ou `-1`, para Fahrenheit, e para executar as funções apropriadas:

```
  switch (scale) {  // Decide por escala Celsius ou Fahrenheit
    case 1:
      celsius(sensor);
      break;
    case -1:
      fahrenheit(sensor);
  }
}
```

Logo a seguir, você tem duas funções para exibir as temperaturas no LCD. Uma trabalha em Celsius, e a outra em Fahrenheit. As funções têm um único parâmetro, um valor inteiro que será lido no sensor de temperatura:

```
void celsius(int sensor) {
```

O cursor é definido em sua posição inicial:

```
lcd.setCursor(0,0);
```

Depois, você pega a leitura do sensor, convertendo-a para graus Celsius e multiplicando o valor por 0,09765625:

```
int temp = sensor * 0.09765625;   // Converte para Celsius
```

Chegamos a esse fator utilizando 100, o alcance do sensor, e dividindo esse número pelo alcance do ADC, 1024:

```
100/1024 = 0,09765625
```

Caso seu sensor tenha um alcance de -40°C a +150°C, o cálculo seria o seguinte (supondo que você tenha um sensor que não produza voltagens negativas):

```
190/1024 = 0,185546875
```

Depois, você imprime esse valor convertido para o LCD, acompanhado do caractere B11011111, o símbolo de grau, seguido por um `C`, para indicar que você está exibindo a temperatura em Celsius:

```
lcd.print(temp);
lcd.write(B11011111);    // Símbolo de grau
lcd.print("C  ");
```

Então, a leitura atual da temperatura é verificada, para ver se ela é maior do que os valores atualmente armazenados de `maxC` e `minC`. Se afirmativo, esses valores são alterados de acordo. Isso manterá um registro ativo da maior e da menor temperatura verificada desde que o Arduino foi ligado:

```
if (temp>maxC) {maxC=temp;}
if (temp<minC) {minC=temp;}
```

Na segunda linha do LCD, você imprime `H` (para HIGH), acompanhado do valor de `maxC`, do símbolo de grau e da letra `C`; e L (para LOW), acompanhado do valor de `minC`, do símbolo de grau e da letra `C`:

```
lcd.setCursor(0,1);
lcd.print("H=");
lcd.print(maxC);
lcd.write(B11011111);
lcd.print("C L=");
lcd.print(minC);
```

```
lcd.write(B11011111);
lcd.print("C  ");
```

A função Fahrenheit faz exatamente o mesmo trabalho, exceto que converte a temperatura em Celsius para Fahrenheit, multiplicando seu valor por 1,8 e adicionando 32:

```
float temp = ((sensor * 0.09765625) * 1.8)+32; // converte para F
```

Agora que você sabe como utilizar um LCD para exibir informações úteis, pode criar seus próprios projetos para exibir dados de sensores, ou criar uma interface de usuário simples.

Resumo

No capítulo 8, você explorou as funções mais populares da biblioteca `LiquidCrystal.h`, e aprendeu como limpar o display, imprimir texto em localizações específicas da tela, tornar o cursor visível, invisível ou fazê-lo piscar, e até mesmo como fazer o texto rolar para a esquerda ou direita. O projeto 24 apresentou uma aplicação simples dessas funções em um sensor de temperatura, apenas um exemplo de como um LCD poderia ser utilizado em um projeto real de exibição de dados.

Assuntos e conceitos abordados no capítulo 8:

- como carregar a biblioteca `LiquidCrystal.h`;
- como conectar um LCD ao Arduino;
- como ajustar o brilho da backlight e o contraste do display, utilizando valores diferentes de resistores;
- como controlar o brilho da backlight a partir de um pino PWM;
- como declarar e inicializar um objeto LiquidCrystal;
- como definir o número correto de colunas e linhas no display;
- como limpar o display LCD utilizando `.clear()`;
- como imprimir a localização do cursor utilizando `.print()`;
- como acender e apagar o display utilizando `.display()` e `.noDisplay()`;
- como definir a localização do cursor utilizando `.setCursor(x, y)`;
- como rolar o display para a esquerda utilizando `.scrollDisplayLeft()`;
- como rolar o display para a direita utilizando `.scrollDisplayRight()`;
- como fazer um cursor visível piscar utilizando `.blink()`;
- como criar caracteres personalizados utilizando `.createChar()`;

- como escrever um único caractere na localização atual do cursor, utilizando `.write()`;
- como funciona um display LCD;
- como ler valores de um sensor de temperatura analógico;
- como elevar a resolução do ADC, utilizando uma referência de voltagem interna;
- como tomar decisões utilizando a instrução switch/case;
- como converter valores ADC para leituras de temperatura em Celsius e Fahrenheit;
- como converter o código para realizar leituras em diferentes sensores de temperatura, com intervalos de valores diferentes.

CAPÍTULO 9

Servomecanismos

Neste capítulo, iniciaremos nossa análise dos servomotores ou servomecanismos. Um servo é um motor com um sistema de feedback, que auxilia no controle da posição do motor. Servos tipicamente giram 180 graus, ainda que você também possa adquirir servos de rotação contínua, ou até mesmo modificar um servo padrão para obter esse efeito. Caso você já tenha tido um aviãozinho de controle remoto, já se deparou com servos; eles são utilizados para controlar as superfícies de voo. Carros de controle remoto utilizam servos no mecanismo de direção, e barcos de controle remoto, para controlar o leme. Da mesma forma, eles são muitas vezes utilizados como as juntas móveis dos braços de pequenos robôs, e para controlar movimentos em animatronics. Talvez até o final deste capítulo você se inspire em colocar alguns servos dentro de um ursinho de pelúcia ou outro brinquedo, dando-lhe movimento. As figuras 9.1 e 9.2 mostram algumas outras formas pelas quais podemos utilizar servos.

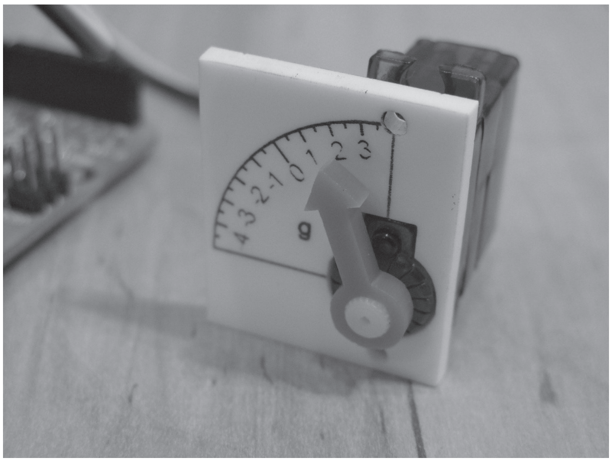


Figura 9.1 – Servo sendo utilizado para controlar um medidor (imagem por Tod E. Kurt).

Graças à biblioteca `Servo.h`, que acompanha o IDE do Arduino, é muito fácil controlar servos. Os três projetos deste capítulo são bem simples e pequenos, quando comparados a alguns dos outros projetos do livro, mas, ainda assim, são muito eficientes. Vamos iniciar com um programa muito simples para controlar um servo, depois passaremos para dois servos, e terminaremos com dois servos controlados por um joystick.

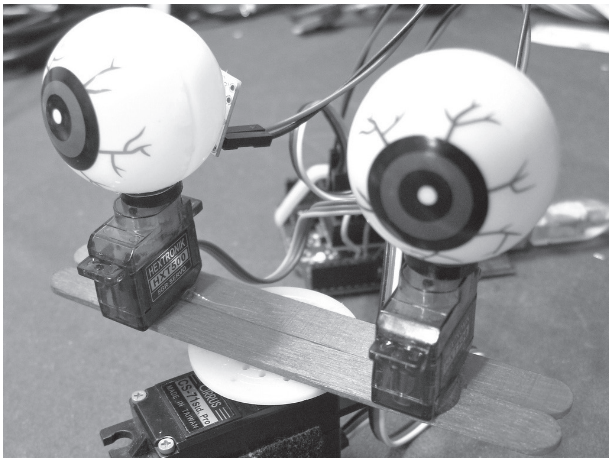


Figura 9.2 – Três servos controlando a cabeça e os olhos de um robô (imagem por Tod E. Kurt).

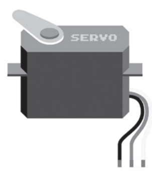
Projeto 25 – Controle de um servo

Neste projeto bem simples, você controlará um único servo utilizando um potenciômetro.

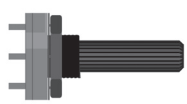
Componentes necessários

Será necessário que você adquira um servo RC padrão; qualquer servo de tamanho pequeno ou médio será suficiente. Servos maiores não são recomendados, pois exigem suas próprias fontes de alimentação, uma vez que consomem muita corrente. Você também necessitará de um potenciômetro; praticamente qualquer potenciômetro rotativo será suficiente. Utilizei um de 4,7 kΩ para teste. Além disso, talvez você queira conectar seu Arduino a uma fonte de alimentação externa.

Servo RC padrão



Potenciômetro rotativo



Conectando os componentes

O circuito para o projeto 25 é muito simples. Faça as conexões como mostra a figura 9.3.

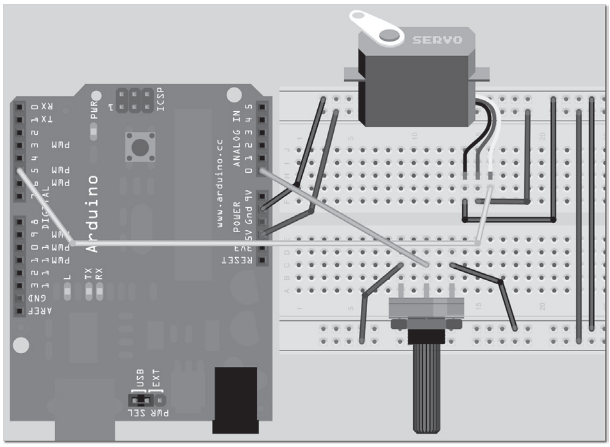


Figura 9.3 – Circuito para o Projeto 25 – Controle de um servo (consulte o site da Novatec para versão colorida).

O servo tem três fios saindo dele. Um será vermelho e irá para os +5 V. Outro, preto ou marrom, irá para o terra. O terceiro, branco, amarelo ou laranja, será conectado ao pino digital 5.

O potenciômetro rotativo tem seus pinos externos conectados ao +5 V e ao terra, e o pino do meio, ao pino analógico 0.

Assim que estiver satisfeito com a forma como tudo foi conectado, digite o código a seguir.

Digite o código

Agora, para ver um dos programas mais curtos do livro, consulte a listagem 9.1.

Listagem 9.1 – Código para o projeto 25

```
// Projeto 25
#include <Servo.h>

Servo servo1;     // Cria um objeto servo

void setup() {
  servo1.attach(5);   // Anexa o servo (físico), no pino 5, ao objeto servo (lógico)
}

void loop() {
  int angle = analogRead(0);       // Lê o valor do potenciômetro
  angle=map(angle, 0, 1023, 0, 180);    // Mapeia os valores de 0 a 180 graus
  servo1.write(angle);         // Escreve o ângulo para o servo
  delay(15);                // Espera de 15ms, para permitir que o servo atinja a posição
}
```

Projeto 25 – Controle de um servo – Análise do código

Primeiramente, incluímos a biblioteca `Servo.h`:

```
#include <Servo.h>
```

Depois, um objeto Servo, `servo1`, é declarado:

```
Servo servo1; // Cria um objeto servo
```

No loop de inicialização, você anexa o servo que acabou de criar ao pino 5:

```
servo1.attach(5); // Anexa o servo (físico), no pino 5, ao objeto servo (lógico)
```

O comando `attach` anexa um objeto servo, criado via programação, a um pino designado. O comando `attach` pode receber um único parâmetro, como no caso presente, ou três parâmetros. Se três parâmetros forem utilizados, o primeiro será o pino, o segundo será o ângulo mínimo (0 grau), medido em largura de pulso, em microssegundos (544, por padrão), e o terceiro parâmetro será o ângulo de grau máximo (180 graus), medido em largura de pulso, também em microssegundos (2400, por padrão). Isso será explicado na análise do hardware. Para a maioria dos projetos, você pode simplesmente definir o pino, e ignorar o segundo e o terceiro parâmetros opcionais.

Você pode conectar até 12 servos a um Arduino Duemilanove (ou equivalente), e até 48 no Arduino Mega — o que é perfeito para aplicações de controle robótico.

Note que o uso dessa biblioteca desabilita a função `analogWrite()` (PWM) nos pinos 9 e 0. No Mega, você pode ter até 12 motores sem interferir com as funções PWM. O uso de 12 a 23 motores desabilitará a funcionalidade PWM nos pinos 11 e 12.

No loop principal, lemos o valor analógico do potenciômetro conectado ao pino analógico 0:

```
int angle = analogRead(0);    // Lê o valor do potenciômetro
```

Depois, esse valor é mapeado de modo que o intervalo, agora, esteja entre `0` e `180`, o que corresponderá ao ângulo de grau do braço do servo:

```
angle=map(angle, 0, 1023, 0, 180);   // Mapeia os valores de 0 a 180 graus
```

Então, você pega o objeto servo e envia a ele o ângulo apropriado, em graus:

```
servo1.write(angle);   // Escreve o ângulo para o servo
```

Por fim, uma espera de 15ms é programada, para dar tempo ao servo, até que alcance a posição desejada:

```
delay(15);    // Espera de 15ms, para permitir que o servo atinja a posição
```

Você também pode utilizar o comando `detach()` para desanexar o servo de um pino, o que desabilitará o servo, permitindo que o pino seja utilizado para algo diferente.

Da mesma forma, você também pode usar o comando `read()` para ler o ângulo atual do servo (ou seja, o último valor passado ao comando `write()`).

Você pode ler mais sobre a biblioteca Servo no site do Arduino, em *http://arduino.cc/en/Reference/Servo*.

Projeto 25 – Controle de um servo – Análise do hardware

Um servo é uma pequena caixa, contendo um motor elétrico CC, um conjunto de engrenagens entre o motor e um eixo de saída, um mecanismo sensor de posição, e um circuito de controle. O mecanismo sensor de posição transmite a posição do servo para o circuito de controle, que utiliza o motor para ajustar o braço do servo na posição que ele deve ocupar.

Servos podem ser encontrados em muitos tamanhos, e em velocidades e precisões diferentes. Alguns podem ser muito caros. Quanto mais poderoso ou preciso for seu servo, maior será o preço. Servos são mais comumente utilizados em aviões, carros e barcos de controle remoto.

A posição de um servo é controlada fornecendo um conjunto de pulsos a ele. Estamos falando novamente de PWM, conceito com o qual você já trabalhou. A largura dos pulsos é medida em milissegundos. A taxa na qual os pulsos são enviados não é particularmente importante; é a largura do pulso que importa para o circuito de controle. Pulsos típicos têm entre 40 Hz e 50 Hz.

Em um servo padrão, a posição central é atingida fornecendo pulsos em intervalos de 1,5 milissegundo; a posição de -45 graus com pulsos de 0,6 milissegundo e a posição de +45 graus com pulsos de 2,4 milissegundos. Você terá de ler o datasheet de seu servo para descobrir as larguras de pulso necessárias para os diferentes ângulos. Entretanto, neste projeto você está utilizando a biblioteca `Servo.h`, por isso não há com o que se preocupar: a própria biblioteca fornece o sinal PWM necessário para o servo. Sempre que você enviar um valor de ângulo diferente para o objeto servo, o código na biblioteca cuidará de enviar o sinal PWM correto.

Alguns servos fornecem rotação contínua. Alternativamente, você também pode modificar um servo padrão, com uma certa facilidade, para obter rotação contínua.

Um servo de rotação contínua é controlado da mesma forma, fornecendo um ângulo entre 0 e 180 graus. Todavia, um valor de 0 fornecerá rotação na velocidade máxima em uma direção, um valor de 90 será estacionário e um valor de 180 fornecerá rotação na velocidade máxima, na direção oposta. Valores entre esses farão com que o servo rotacione em uma direção ou outra, a velocidades diferentes. Servos de rotação contínua são ótimos para construção de pequenos robôs (Figura 9.4). Eles podem ser conectados a rodas, para fornecer controle preciso de velocidade e direção sobre cada uma delas.

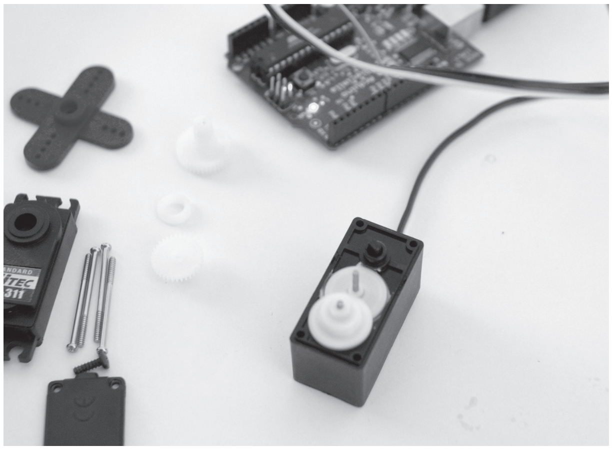


Figura 9.4 – Modificação de servo para obter rotação contínua (imagem por Adam Grieg).

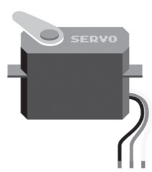
Há outro tipo de servo, o atuador linear, que rotaciona um eixo até uma posição desejada, permitindo que você empurre ou puxe itens conectados à extremidade do eixo. Eles são muito utilizados no programa de TV "Mythbusters" ("Os Caçadores de Mitos") pelo especialista em robótica da equipe do programa, Grant Imahara.

Projeto 26 – Controle de um servo duplo

Agora você criará outro projeto simples, mas, dessa vez, controlará dois servos utilizando comandos do monitor serial. Você aprendeu sobre controle serial no projeto 10, quando alterou as cores de uma lâmpada RGB utilizando comandos seriais. Por isso, vamos pegar o que pudermos do código do projeto 10 e criar o código que utilizaremos neste projeto.

Componentes necessários

2 servos RC padrão



Conectando os componentes

O circuito para o projeto 26 é, uma vez mais, muito simples. Conecte os componentes como na figura 9.5. Basicamente, basta remover o potenciômetro do último projeto e ligar um segundo servo no pino digital 6.

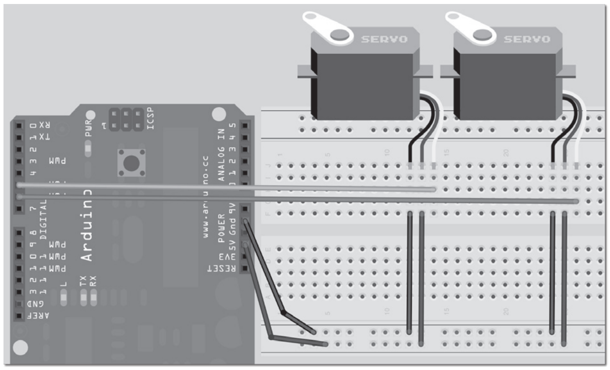


Figura 9.5 – Circuito para o Projeto 26 – Controle de um servo duplo (consulte o site da Novatec para versão colorida).

Digite o código

Digite o código da listagem 9.2.

Listagem 9.2 – Código para o projeto 26

```
// Projeto 26
#include <Servo.h>

char buffer[10];
Servo servo1;   // Cria um objeto servo
Servo servo2;   // Cria um segundo objeto servo

void setup() {
  servo1.attach(5);    // Anexa o servo, no pino 5, ao objeto servo1
  servo2.attach(6);    // Anexa o servo, no pino 6, ao objeto servo2
  Serial.begin(9600);
  Serial.flush();
  servo1.write(90);    // Coloca o servo1 na posição inicial
  servo2.write(90);    // Coloca o servo2 na posição inicial
  Serial.println("STARTING...");
}

void loop() {
  if (Serial.available() > 0) {  // Verifica se dados foram digitados
    int index=0;
    delay(100);   // Deixa o buffer encher
    int numChar = Serial.available(); // Encontra o comprimento da string
    if (numChar>10) {
      numChar=10;
    }
```

```
      while (numChar--) {
        // Preenche o buffer com a string
        buffer[index++] = Serial.read();
      }
      splitString(buffer);   // Executa a função splitString
    }
  }

  void splitString(char* data) {
    Serial.print("Data entered: ");
    Serial.println(data);
    char* parameter;
    parameter = strtok (data, " ,");    // De string para tokens
    while (parameter != NULL) {       // Se ainda não atingimos o fim da string...
      setServo(parameter);       // ...execute a função setServo
      parameter = strtok (NULL, " ,");
    }
    // Limpa o texto e os buffers seriais
    for (int x=0; x<9; x++) {
      buffer[x]='\0';
    }
    Serial.flush();
  }

  void setServo(char* data) {
    if ((data[0] == 'L') || (data[0] == 'l')) {
      int firstVal = strtol(data+1, NULL, 10);    // De string para inteiro longo
      firstVal = constrain(firstVal,0,180);   // Restringe os valores
      servo1.write(firstVal);
      Serial.print("Servo1 is set to: ");
      Serial.println(firstVal);
    }
    if ((data[0] == 'R') || (data[0] == 'r')) {
      int secondVal = strtol(data+1, NULL, 10);  // De string para inteiro longo
      secondVal = constrain(secondVal,0,255);    // Restringe os valores
      servo2.write(secondVal);
      Serial.print("Servo2 is set to: ");
      Serial.println(secondVal);
    }
  }
```

Para executar o código, abra a janela do monitor serial. O Arduino reinicializará, e os servos alcançarão suas posições centrais. Você, agora, pode utilizar o monitor serial para enviar comandos ao Arduino.

O servo da esquerda é controlado enviando-se um L, acompanhado de um número entre 0 e 180 para o ângulo. O servo da direita é controlado enviando-se um R, acompanhado de um número nesse mesmo intervalo. Você pode enviar comandos individuais para cada servo, ou enviar comandos aos dois servos, ao mesmo tempo, separando-os por um espaço ou uma vírgula, da seguinte maneira:

```
L180
L45 R135
L180,R90
R77
R25 L175
```

Esse é um exemplo simples de como você pode enviar comandos para um braço de robô controlado pelo Arduino, ou para um brinquedo de animatronic. Note que os comandos seriais não têm necessariamente de vir do Serial Monitor do Arduino. Você pode utilizar qualquer programa capaz de se comunicar de forma serial, ou até mesmo escrever o seu programa em Python ou C++.

Projeto 26 – Controle de um servo duplo – Análise do código

O código para este projeto é basicamente uma repetição do código do projeto 10. Portanto, não analisarei cada comando detalhadamente. Em vez disso, apresentarei uma visão geral, quando falarmos de algo que já abordamos. Leia novamente o projeto 10, para relembrar como funciona a manipulação de strings.

Primeiramente, incluímos a biblioteca `Servo.h`:

```
#include <Servo.h>
```

Depois, um array de tipo `char` é criado para armazenar a string de texto que você digitou como comando no monitor serial:

```
char buffer[10];
```

Dois objetos servo são criados:

```
Servo servo1;   // Cria um objeto servo
Servo servo2;   // Cria um segundo objeto servo
```

Na rotina de inicialização, anexe os objetos servos aos pinos 5 e 6:

```
servo1.attach(5);    // Anexa o servo, no pino 5, ao objeto servo1
servo2.attach(6);    // Anexa o servo, no pino 6, ao objeto servo2
```

Depois, inicie as comunicações seriais e execute um comando `Serial.flush()` para limpar os caracteres do buffer serial, deixando-o vazio e pronto para receber comandos que serão enviados aos servos:

```
Serial.begin(9600);
Serial.flush();
```

Escrevemos para ambos os servos um valor de `90`, que corresponde ao ponto central, de modo que iniciem na posição central:

```
servo1.write(90);   // Coloca o servo1 na posição inicial
servo2.write(90);   // Coloca o servo2 na posição inicial
```

Então a palavra "`STARTING...`" é exibida na janela do monitor serial, para que você saiba que o dispositivo está pronto para receber comandos:

```
Serial.println("STARTING...");
```

No loop principal, verifique se dados foram enviados pela linha de comunicação serial:

```
if (Serial.available() > 0) { // Verifica se dados foram digitados
```

Se afirmativo, deixe que o buffer encha e obtenha o comprimento da string, garantindo que ele não ultrapasse o máximo de dez caracteres. Assim que o buffer estiver cheio, chame a rotina `splitString`, enviando o array de buffer à função:

```
int index=0;
delay(100);  // Deixa o buffer encher
int numChar = Serial.available();  // Encontra o comprimento da string
if (numChar>10) {
  numChar=10;
}
while (numChar--) {
  // Preenche o buffer com a string
  buffer[index++] = Serial.read();
}
splitString(buffer);  // Executa a função splitString
```

A função `splitString` recebe o array de buffer, faz sua divisão em comandos separados, caso tenha sido digitado mais de um, e chama a rotina `setServo`, com o parâmetro retirado da string de comando recebida pela linha serial:

```
void splitString(char* data) {
  Serial.print("Data entered: ");
  Serial.println(data);
  char* parameter;
  parameter = strtok (data, " ,");   // De string para tokens
  while (parameter != NULL) {      // Se ainda não atingimos o fim da string...
    setServo(parameter);      // ...execute a função setServo
    parameter = strtok (NULL, " ,");
  }
  // Limpa o texto e os buffers seriais
  for (int x=0; x<9; x++) {
    buffer[x]='\0';
  }
  Serial.flush();
}
```

A rotina `setServo` recebe a menor string enviada pela função `splitString`, e verifica se um `L` ou `R` foi digitado. Se afirmativo, ela movimenta o servo esquerdo ou direito pelo valor especificado na string:

```
void setServo(char* data) {
  if ((data[0] == 'L') || (data[0] == 'l')) {
    int firstVal = strtol(data+1, NULL, 10);  // De string para inteiro longo
    firstVal = constrain(firstVal,0,180);     // Restringe os valores
    servo1.write(firstVal);
    Serial.print("Servo1 is set to: ");
    Serial.println(firstVal);
  }
  if ((data[0] == 'R') || (data[0] == 'r')) {
    int secondVal = strtol(data+1, NULL, 10);  // De string para inteiro longo
    secondVal = constrain(secondVal,0,255);    // Restringe os valores
    servo2.write(secondVal);
    Serial.print("Servo2 is set to: ");
    Serial.println(secondVal);
  }
```

Abordei rapidamente essas últimas funções, uma vez que elas são praticamente idênticas àquelas do projeto 10. Se você não se lembra do que abordamos naquele projeto, sinta-se livre para lê-lo novamente.

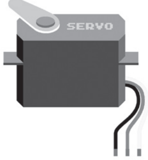
Projeto 27 – Controle de servos com joystick

Para outro projeto simples, vamos utilizar um joystick para controlar os dois servos. Você posicionará os servos, de modo a obter controle panorâmico e de inclinação (*pan/tilt*) semelhante ao de uma cabeça humana, como o que temos em câmeras de circuito fechado ou em sensores de montagem em robôs.

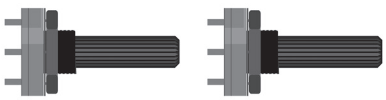
Componentes necessários

Deixe o circuito como o do último projeto, adicionando apenas dois potenciômetros, ou um joystick potenciômetro de dois eixos.

2 servos RC padrão



Joystick potenciômetro de dois eixos (ou dois potenciômetros)



Conectando os componentes

O circuito para o projeto 27 é o mesmo do projeto 26, com a adição do joystick.

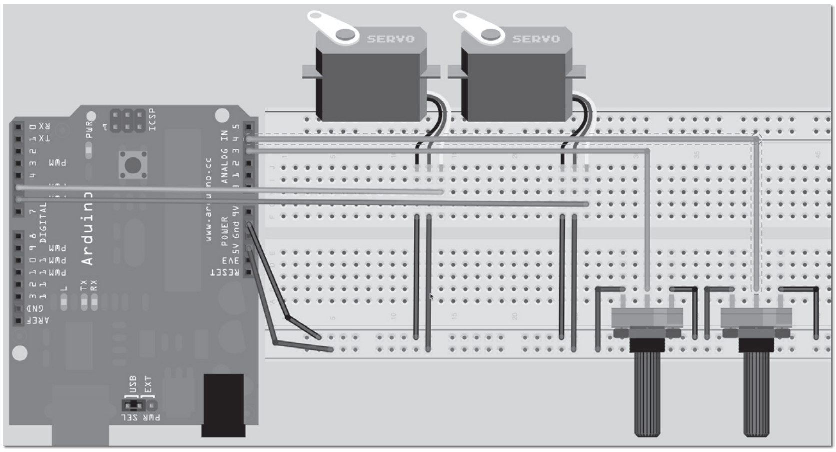


Figura 9.6 – Circuito para o Projeto 27 – Controle de servos com joystick (consulte o site da Novatec para versão colorida).

Um joystick potenciômetro é simplesmente o que indica seu nome: um joystick formado de dois potenciômetros, posicionados em ângulos retos. Os eixos dos potenciômetros são conectados a uma barra que pode ser movimentada em dois sentidos, utilizando o joystick, e que retorna à sua posição central, graças a um conjunto de molas.

A conexão, portanto, é simples: os pinos externos dos dois potenciômetros vão para o +5 V e para o terra, e os pinos centrais, para os pinos analógicos 3 e 4. Caso você não tenha um joystick, dois potenciômetros dispostos a 90 graus um do outro podem ser suficientes.

Conecte os dois servos, de forma que um tenha seu eixo na vertical e o outro, a 90 graus do primeiro servo, tenha seu eixo na horizontal e preso lateralmente à estrutura do primeiro servo. Consulte a figura 9.7 para ver como conectar os servos. Um pouco de cola quente será suficiente para nossos testes. Utilize uma cola mais forte para fixação permanente.

Como alternativa, adquira um dos kits prontos de servos com função pan/tilt, que podem ser comprados para projetos de robótica. Kits desse tipo podem ser comprados a preços baixos no eBay.

Quando o servo da base se movimenta, ele faz com que o servo do topo rotacione, e quando o servo do topo se movimenta, seu braço balança para frente e para trás. Você poderia anexar uma webcam ou sensores ultrassônicos ao braço, por exemplo.

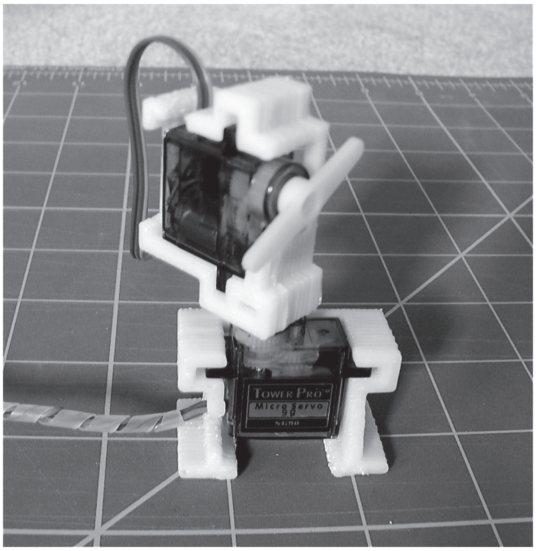


Figura 9.7 – Monte um servo sobre o outro (imagem por David Stokes).

O joystick pode ser adquirido no eBay ou em um fornecedor de materiais elétricos. Você também poderia encontrar um velho Atari ou C64 e utilizar seus joysticks. Entretanto, há uma alternativa muito barata disponível: um controle de PlayStation 2 (PS2), com dois potenciômetros de dois eixos, além de motores de vibração e outros botões. Joysticks desse tipo podem ser adquiridos no eBay a preços muito baixos, e são facilmente desmontados para que tenhamos acesso a seus componentes (Figura 9.8). Caso você não queira desmontar o controle, pode acessar o código digital vindo do cabo do controle de PS2. De fato, há bibliotecas do Arduino que permitem a você fazer exatamente isso. Dessa forma, você terá acesso imediato a todos os joysticks e botões do dispositivo.

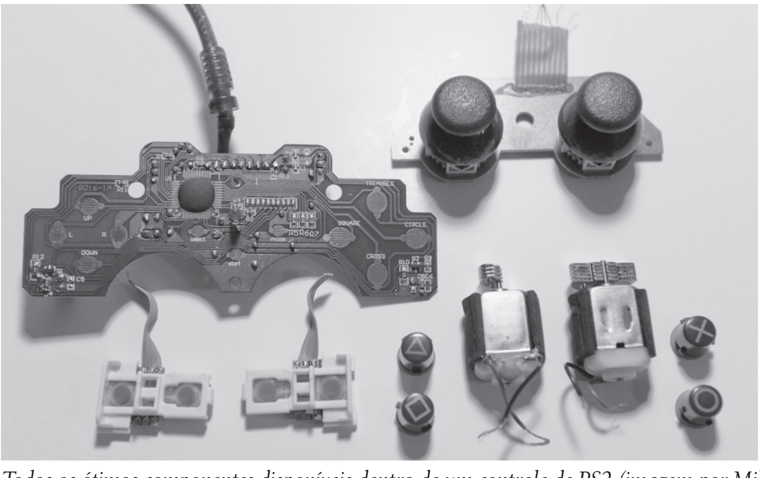


Figura 9.8 – Todos os ótimos componentes disponíveis dentro de um controle de PS2 (imagem por Mike Prevette).

Digite o código

Digite o código na listagem 9.3.

Listagem 9.3 – Código para o projeto 27

```
#include <Servo.h>

Servo servo1;   // Cria um objeto servo
Servo servo2;   // Cria um segundo objeto servo
int pot1, pot2;

void setup() {
  servo1.attach(5);   // Anexa o servo, no pino 5, ao objeto servo1
  servo2.attach(6);   // Anexa o servo, no pino 6, ao objeto servo2

  servo1.write(90);   // Coloca o servo1 na posição inicial
  servo2.write(90);   // Coloca o servo2 na posição inicial
}

void loop() {
  pot1 = analogRead(3);    // Lê o eixo X
  pot2 = analogRead(4);    // Lê o eixo Y
  pot1 = map(pot1,0,1023,0,180);
  pot2=map(pot2,0,1023,0,180);
  servo1.write(pot1);
  servo2.write(pot2);
  delay(15);
}
```

Quando você executar este programa, será capaz de utilizar os servos para realizar movimentos de tipo pan/tilt. Mover o joystick para frente e para trás fará com que a estrutura do servo do topo também se movimente nesse sentido, enquanto mover o joystick lateralmente fará com que o servo da base rotacione.

Se os servos estiverem se movendo na direção oposta à esperada, você deve ter conectado os pinos ao contrário. Apenas inverta as conexões.

Projeto 27 – Controle de servos com joystick – Análise do código

Novamente, este é um projeto bem simples, mas o efeito da movimentação dos dois servos é muito útil.

Primeiramente, carregamos a biblioteca Servo:

```
#include <Servo.h>
```

Dois objetos servo são criados, e duas variáveis de tipo inteiro armazenam os valores lidos a partir dos dois potenciômetros, dentro do joystick:

```
Servo servo1;   // Cria um objeto servo
```

```
Servo servo2;   // Cria um segundo objeto servo
int pot1, pot2;
```

O loop de inicialização anexa os dois objetos servo aos pinos 5 e 6, e movimenta os servos até que atinjam suas posições centrais:

```
servo1.attach(5);    // Anexa o servo, no pino 5, ao objeto servo1
servo2.attach(6);    // Anexa o servo, no pino 6, ao objeto servo2

servo1.write(90);    // Coloca o servo1 na posição inicial
servo2.write(90);    // Coloca o servo2 na posição inicial
```

No loop principal, os valores analógicos são lidos a partir dos eixos X e Y do joystick:

```
pot1 = analogRead(3);     // Lê o eixo X
pot2 = analogRead(4);     // Lê o eixo Y
```

Esses valores são, então, mapeados para graus entre `0` e `180`:

```
pot1 = map(pot1,0,1023,0,180);
pot2 = map(pot2,0,1023,0,180);
```

E enviados aos dois servos:

```
servo1.write(pot1);
servo2.write(pot2);
```

O alcance do movimento disponível com esse equipamento de pan/tilt é incrível, e você pode fazer com que ele se mova de forma muito humana. Esse tipo de configuração de servos é, muitas vezes, utilizado para controlar uma câmera de fotografia aérea.

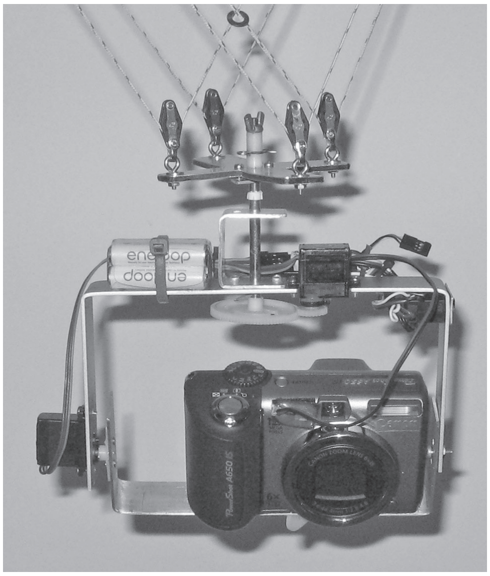


Figura 9.9 – Equipamento com capacidade pan/tilt, feito para uma câmera utilizando dois servos (imagem por David Mitchell).

Resumo

No capítulo 9, você trabalhou com três projetos muito simples, que serviram para mostrar como podemos facilmente controlar servos utilizando a biblioteca `Servo.h`. Você pode, se desejar, modificar esses projetos e adicionar até 12 servos para, por exemplo, fazer com que um dinossauro de brinquedo se movimente de modo realístico.

Servos são ótimos para criar seu próprio carrinho de controle remoto, ou para controlar um robô. Além disso, como vimos na figura 9.9, você também pode criar um equipamento com capacidade pan/tilt, para controle de uma câmera. Um terceiro servo poderia ser utilizado para apertar o botão do obturador na câmera.

Assuntos e conceitos abordados no capítulo 9:

- os vários usos em potencial de um servo;
- como utilizar a biblioteca `Servo.h` para controlar de 1 a 12 servos;
- utilização de um potenciômetro como controlador de um servo;
- como funciona um servo;
- como modificar um servo para fornecer rotação contínua;
- como controlar um conjunto de servos utilizando comandos seriais;
- como utilizar um joystick analógico para controlar um servo em dois eixos;
- como posicionar dois servos para criar um movimento de pan/tilt, como em uma cabeça humana;
- como um controle de PS2 é uma ótima fonte de componentes para joysticks e botões.

CAPÍTULO 10

Motores de passo e robôs

Agora você conhecerá um novo tipo de motor, o motor de passo. O projeto 28 é um projeto simples, que mostra como controlar um motor de passo, fazendo-o se movimentar a uma distância definida, além de alterar sua velocidade e direção. Motores de passo são diferentes de motores padrão: sua rotação é dividida em uma série de passos. Fazendo com que o motor rotacione um número definido de passos, você pode controlar sua velocidade e seus giros com muita precisão. Motores de passo podem ser encontrados em muitos formatos e tamanhos diferentes, tendo quatro, cinco ou seis fios.

Motores de passo têm muitas utilidades; são usados em scanners de mesa para posicionar a cabeça de leitura, e em impressoras jato de tinta para controlar a posição da cabeça e do papel.

Outro projeto, neste capítulo, fará com que você utilize um shield de motor, com motores CC engrenados para controlar uma base de robô. Ao final, você conseguirá fazer com que um robô se movimente, acompanhando uma linha preta desenhada no chão.

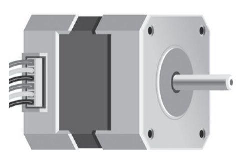
Projeto 28 – Controle básico de um motor de passo

Neste projeto bem simples, você conectará um motor de passo e fará com que ele seja controlado pelo Arduino, utilizando direções e velocidades diferentes para um número definido de passos.

Componentes necessários

Você necessitará de um motor de passo CC bipolar ou unipolar. Em meu caso, utilizei um motor de passo unipolar Sanyo 103H546-0440 nos testes deste projeto.

Motor de passo



CI controlador de motor L293D ou SN754410



2 Capacitores cerâmicos de 0,01 µF



Resistor limitador de corrente



Conectando os componentes

Conecte tudo como na figura 10.1. Veja a variação para motores bipolares.

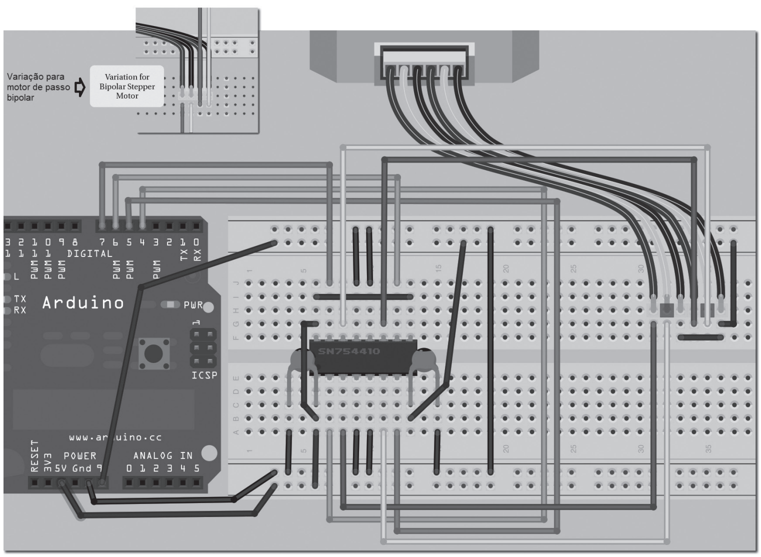


Figura 10.1 – Circuito para o Projeto 28 – Controle básico de um motor de passo (consulte o site da Novatec para versão colorida).

Certifique-se de que seu Arduino esteja conectado a uma fonte de alimentação externa, para não sobrecarregá-lo. Os capacitores são opcionais, e ajudam a suavizar a corrente e impedir interferências com o Arduino. Esses capacitores devem ser posicionados entre o +5 V e o terra, e entre a fonte de alimentação e o terra. Você também pode utilizar capacitores eletrolíticos de baixo valor, mas certifique-se de conectá-los corretamente, uma vez que são polarizados.

Também pode ser necessário um resistor limitador de corrente entre o pino Vin no Arduino e o barramento de alimentação que alimenta o chip SN754410 ou L293D. O pino Vin fornecerá qualquer voltagem advinda de uma fonte de alimentação externa,

por isso você pode ter de diminuir essa voltagem, ajustando-a ao valor que o motor requer. Se não fizer isso, você pode acabar danificando o motor.

Os pinos digitais 4, 5, 6 e 7 do Arduino vão para os pinos de entrada (Input) 1, 2, 3 e 4 do controlador de motor (pinos 2, 7, 10 e 15, respectivamente; veja a Figura 10.2). Os pinos de saída 1 e 2 do controlador de motor vão para a bobina 1 do motor, e os pinos de saída 3 e 4 vão para a bobina 2. Você terá de verificar o datasheet de seu motor específico, para descobrir quais fios coloridos vão para a bobina 1 e quais vão para a bobina 2. Um motor unipolar também terá um quinto e/ou um sexto fio indo para o terra.

O pino de 5 V do Arduino vai para o pino 16 (VSS) do controlador de motor, e os dois pinos inibidores do chip (1 e 9) também estão vinculados à linha de 5 V, para que tenham valor `HIGH`.

O pino Vin (9 V) do Arduino vai para o pino 8 do CI controlador (VC). Os pinos 4, 5, 12 e 13 vão para o terra.

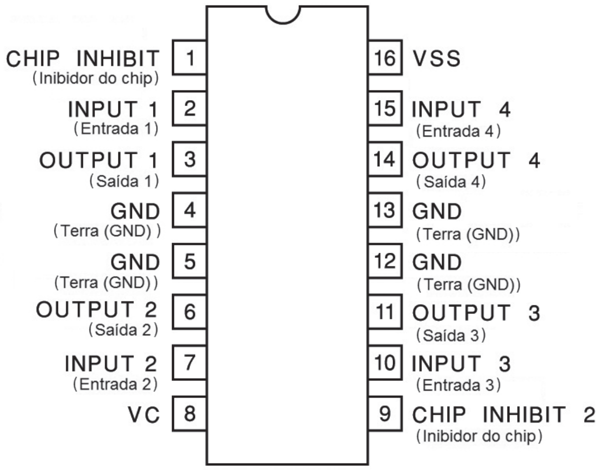


Figura 10.2 – Diagrama de pinos para um CI controlador de motor L293D ou SN754410.

Assim que você estiver seguro de que tudo foi conectado da forma correta, digite o código a seguir.

Digite o código

Digite o código da listagem 10.1.

Listagem 10.1 – Código para o projeto 28

```
// Projeto 28
#include <Stepper.h>
```

```
// o valor de um passo é 360 / ângulo de grau do motor
#define STEPS 200

// crie um objeto stepper nos pinos 4, 5, 6 e 7
Stepper stepper(STEPS, 4, 5, 6, 7);

void setup() {
}

void loop() {
  stepper.setSpeed(60);
  stepper.step(200);
  delay(100);
  stepper.setSpeed(20);
  stepper.step(-50);
  delay(100);
}
```

Certifique-se de que seu Arduino esteja conectado a uma fonte de alimentação externa antes de executar o código. Quando o sketch for executado, você verá o motor de passo realizar uma rotação completa, parar por um instante, rotacionar no sentido contrário por um quarto de rotação completa, parar por um instante e, então, repetir o processo. Pode ser interessante prender uma tira de fita ao eixo do motor, para que você possa observá-lo rotacionar.

Projeto 28 – Controle básico de um motor de passo – Análise do código

O código para este projeto é simples e direto, graças à biblioteca `Stepper.h`, que cuida de todo o trabalho para nós. Primeiramente, você inclui a biblioteca no sketch:

```
#include <Stepper.h>
```

Depois, você tem de definir quantos passos são necessários para que o motor realize uma rotação completa de 360 graus. Tipicamente, motores de passo vêm em variedades de 7,5 ou 1,8 graus, mas você pode ter um motor de passo com um ângulo de passo diferente. Para calcular os passos basta dividir 360 pelo ângulo de passo. No caso do motor de passo que utilizei, o ângulo de passo era 1,8 grau, significando que 200 passos são necessários para executar uma rotação completa de 360 graus.

```
#define STEPS 200
```

Em seguida, você cria um objeto para o motor de passo, dá a ele o nome `stepper` e atribui os pinos relacionados a cada lado das duas bobinas:

```
Stepper stepper(STEPS, 4, 5, 6, 7);
```

A função de inicialização não faz nada, mas tem de ser incluída mesmo assim:

```
void setup() {
}
```

No loop principal, você primeiro define a velocidade do motor em rpm (rotações por minuto). Isso é feito com o comando `.setSpeed()`, e a velocidade, em rpm, é incluída dentro dos parênteses:

```
stepper.setSpeed(60);
```

Em seguida, você diz ao motor de passo quantos passos ele deve executar. Aqui, definimos 200 passos a 60 rpm, o que significa que ele executará uma revolução completa a cada segundo:

```
stepper.step(200);
```

Ao final dessa revolução, uma espera de um décimo de segundo é implementada:

```
delay(100);
```

Então, a velocidade é diminuída para apenas 20 rpm:

```
stepper.setSpeed(20);
```

Na sequência, fazemos com que o motor rotacione na direção oposta (daí o valor negativo) por 50 passos, o que, em um motor de 200 passos, representa um quarto de rotação:

```
stepper.step(-50);
```

Então, esperamos mais cem milissegundos, antes que o loop se repita.

Como você pode ver, o controle de um motor de passo, mesmo sendo muito simples, permite que você controle sua velocidade, direção e exatamente quanto o eixo vai rotacionar. Dessa forma, você pode controlar exatamente até onde o item preso ao eixo rotacionará, e por quantos graus. Utilizar um motor de passo para controlar a roda de um robô resultaria em um controle muito preciso. Da mesma forma, utilizá-lo para rotacionar a cabeça de um robô, ou uma câmera, permitiria que você posicionasse o elemento anexado ao motor exatamente no ponto em que deseja.

Projeto 28 – Controle básico de um motor de passo – Análise do hardware

O novo elemento de hardware que você está utilizando neste projeto é o motor de passo. Um motor de passo é um motor CC no qual a rotação pode ser dividida em passos menores. Isso é feito utilizando um rotor de ferro, com forma de engrenagem, anexado aos eixos dentro do motor. Em volta da parte externa do motor temos eletromagnetos dentados. Uma bobina é energizada, fazendo com que os dentes do rotor de ferro se alinhem aos dentes do eletromagneto. Os dentes do próximo eletromagneto ficam levemente deslocados em relação aos primeiros; quando ele é o eletromagneto energizado, e a primeira bobina é desligada, isso faz com que o eixo rotacione um pouco mais em direção ao próximo eletromagneto. Esse processo se

repete pelo número de eletromagnetos que existem dentro do motor, até que os dentes estejam praticamente alinhados com o primeiro eletromagneto, repetindo o processo.

Cada vez que um eletromagneto é energizado e o rotor se move um pouco, ele executa um passo. Invertendo a sequência dos eletromagnetos que energizam o rotor, ele gira na direção oposta.

O trabalho do Arduino é aplicar os comandos `HIGH` e `LOW` apropriados às bobinas, na sequência e velocidade corretas, para permitir que o eixo rotacione. Isso é o que ocorre nos bastidores com a biblioteca `Stepper.h`.

Um motor unipolar tem cinco ou seis fios que vão para quatro bobinas. Os pinos do centro, nas bobinas, ficam presos e vão para a fonte de alimentação. Motores bipolares geralmente têm quatro fios e nenhuma conexão comum no centro. Na figura 10.3, mostramos os diferentes tipos e tamanhos de motores de passo.



Figura 10.3 – Tipos diferentes de motores de passo (imagem por Aki Korhonen).

A sequência de passos de um motor de passos pode ser vista na tabela 10.1.

Tabela 10.1 – Sequência de passos de um motor de passo

Passo	Fio 1	Fio 2	Fio 4	Fio 5
1	HIGH	LOW	HIGH	LOW
2	LOW	HIGH	HIGH	LOW
3	LOW	HIGH	LOW	HIGH
4	HIGH	LOW	LOW	HIGH

A figura 10.4 mostra o diagrama de um motor de passo unipolar. Como você pode ver, há quatro bobinas (na verdade duas, mas elas estão divididas pelos fios do centro em duas bobinas menores). O fio do centro de cada bobina vai para a fonte de alimentação; os dois outros fios da primeira bobina vão para as saídas em um dos lados da ponte H do CI controlador, e os outros dois da segunda bobina vão para as duas últimas saídas da ponte H.

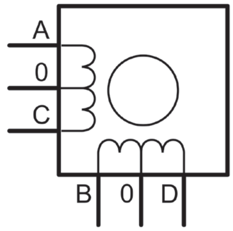


Figura 10.4 – Diagrama de circuito para um motor de passo unipolar.

A figura 10.5 mostra um diagrama para um motor de passo bipolar. Dessa vez, há apenas duas bobinas, e nenhum pino central. A sequência de passos para um motor bipolar é a mesma de um motor unipolar. No entanto, dessa vez você inverte a corrente que atravessa as bobinas, ou seja, um `HIGH` significa uma voltagem positiva, e um `LOW` significa o terra (ou uma conexão direcionada para o estado baixo).

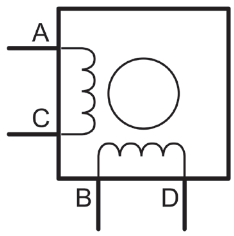


Figura 10.5 – Diagrama de circuito para um motor de passo bipolar.

É possível aumentar a resolução do motor de passo, utilizando técnicas especiais que empregam meio-passos ou micropassos. Essas alternativas são capazes de aumentar a resolução do motor, mas ao custo de uma confiabilidade reduzida de posição e torque.

Agora que você sabe tudo sobre motores de passo, voltaremos a controlar um motor normalmente, mas dessa vez utilizando um shield de motor e controlando dois motores, conectados a rodas em uma base de robô.

Projeto 29 – Uso de um shield de motor

Agora você utilizará um shield de motor para controlar dois motores separadamente. Você aprenderá como controlar a velocidade e a direção de cada motor, e como aplicar esse conhecimento para controlar as rodas de um robô de duas rodas. Quando avançar ao projeto 30, essas habilidades serão empregadas na criação de um robô que acompanha uma linha desenhada no chão.

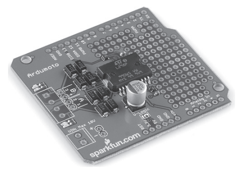
Componentes necessários

Você pode executar este projeto utilizando dois motores CC, se isso for tudo que estiver disponível. Entretanto, essa opção não é muito divertida. Por isso, será melhor adquirir uma base de robô com duas rodas, ou criar uma a partir de dois motores

engrenados com rodas. A rotação a meia velocidade deve resultar em cerca de 20 cm, ou 6 polegadas, de movimento para frente por segundo.

A fonte de alimentação terá de oferecer tensão suficiente para alimentar o Arduino, o shield e os motores. Utilizei seis pilhas AA em um suporte, além de um plugue fêmea soldado, para alimentar minha configuração nos testes deste projeto.

Shield de motor



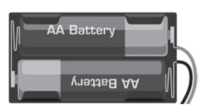
2 motores CC ou...



... uma base de robô com duas rodas



Fonte de alimentação



Conectando os componentes

Como este projeto não requer nada além de um shield de motor, conectado ao Arduino, e os fios de seus dois motores, conectados às quatro saídas do shield, não é necessário um diagrama de circuito. Em vez disso, na figura 10.6, temos uma bela imagem de um shield de motor conectado a um Arduino.

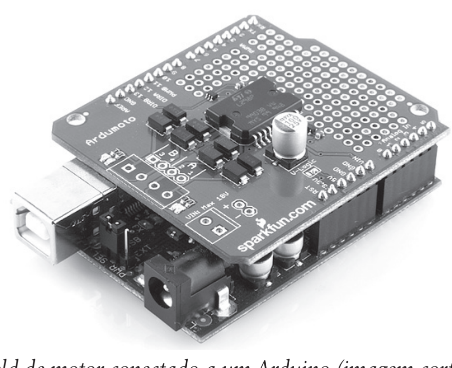


Figura 10.6 – Shield de motor conectado a um Arduino (imagem cortesia da Sparkfun).

Para os testes deste projeto, utilizei um shield de motor Ardumoto da Sparkfun, projetado para controlar dois motores. Também há um shield disponível pela Adafruit, que permite controlar até quatro motores CC, dois motores de passo ou dois servos. Além disso, ele também vem acompanhado da excelente biblioteca `AFMotor.h`, fazendo com que seja fácil controlar motores, motores de passo ou servos. A escolha é sua. Todavia, o código para este projeto foi elaborado considerando o Ardumoto, por isso, caso você venha a utilizar um shield diferente, será necessário modificar o código de acordo.

Cada motor terá de ser conectado através das portas A (saídas 1 e 2) e B (saídas 3 e 4) do shield.

Também utilizei uma base de robô de duas rodas da DFRobot (Figura 10.7); trata-se de uma boa base de baixo custo, que vem acompanhada de anexos e de dois furos de montagem para encaixe de sensores. Ela é ideal para o projeto de robô em que trabalharemos a seguir. Entretanto, qualquer base de robô será suficiente para este projeto; você poderia até mesmo utilizar uma base de robô de quatro rodas, desde que não se esquecesse de controlar ambos os conjuntos de motores em cada lado, em vez de apenas um. Evidentemente, se você não quer utilizar uma base de robô, pode realizar esse teste com dois motores CC.

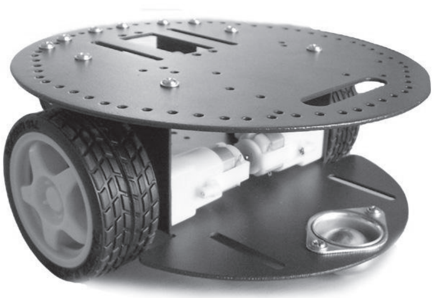


Figura 10.7 – Base de robô de duas rodas (imagem cortesia da DFRobot).

Digite o código

Digite o código da listagem 10.2.

Listagem 10.2 – Código para o projeto 29

```
// Projeto 29 – Uso de um shield de motor

// Define os pinos de velocidade e direção de cada motor
int speed1 = 3;
int speed2 = 11;
int direction1 = 12;
int direction2 = 13;
```

```
void stopMotor() {
  // desliga ambos os motores
  analogWrite(speed1, 0);
  analogWrite(speed2, 0);
}

void setup() {
  // define todos os pinos como saídas
  pinMode(speed1, OUTPUT);
  pinMode(speed2, OUTPUT);
  pinMode(direction1, OUTPUT);
  pinMode(direction2, OUTPUT);
}

void loop() {
  // Ambos os motores para frente, com 50% da velocidade, por 2 segundos
  digitalWrite(direction1, HIGH);
  digitalWrite(direction2, HIGH);
  analogWrite(speed1,128);
  analogWrite(speed2,128);
  delay(2000);

  stopMotor(); delay(1000); // para

  // Vira para a esquerda por 1 segundo
  digitalWrite(direction1, LOW);
  digitalWrite(direction2, HIGH);
  analogWrite(speed1, 128);
  analogWrite(speed2, 128);
  delay(1000);

  stopMotor(); delay(1000); // para

  // Ambos os motores para frente, com 50% da velocidade, por 2 segundos
  digitalWrite(direction1, HIGH);
  digitalWrite(direction2, HIGH);
  analogWrite(speed1,128);
  analogWrite(speed2,128);
  delay(2000);

  stopMotor(); delay(1000); // para

  // Vira para a direita com 25% da velocidade
  digitalWrite(direction1, HIGH);
  digitalWrite(direction2, LOW);
  analogWrite(speed1, 64);
```

```
  analogWrite(speed2, 64);
  delay(2000);

  stopMotor(); delay(1000); // para
}
```

Projeto 29 – Uso de um shield de motor – Análise do código

O primeiro elemento de destaque é a definição de quais pinos devem controlar a velocidade e a direção. No shield Ardumoto, esses são os pinos 3, 11, 12 e 13.

```
int speed1 = 3;
int speed2 = 11;
int direction1 = 12;
int direction2 = 13;
```

Depois, crie uma função para desligar os motores. O motor é desligado quatro vezes no código, por isso faz sentido criar uma função com esse propósito:

```
void stopMotor() {
  // desliga ambos os motores
  analogWrite(speed1, 0);
  analogWrite(speed2, 0);
}
```

Para desligar os motores, você tem apenas de definir a velocidade de cada motor como zero. Dessa forma, você escreve um valor de zero para os pinos `speed1` (motor da esquerda) e `speed2` (motor da direita).

No loop de incialização, os quatros pinos são definidos em modo de saída:

```
pinMode(speed1, OUTPUT);
pinMode(speed2, OUTPUT);
pinMode(direction1, OUTPUT);
pinMode(direction2, OUTPUT);
```

No loop principal, você executa quatro rotinas de movimento separadas. Primeiro, você avança o robô a 50% de sua velocidade máxima por dois segundos:

```
// Ambos os motores para frente, com 50% da velocidade, por 2 segundos
digitalWrite(direction1, HIGH);
digitalWrite(direction2, HIGH);
analogWrite(speed1,128);
analogWrite(speed2,128);
delay(2000);
```

Isso é feito, primeiramente, definindo os pinos de direção como `HIGH`, o que significa que eles devem avançar (desde que seus motores estejam com a fiação correta). Os pinos de velocidade de ambos os motores são definidos como `128`. Os valores PWM

têm alcance de 0 a 255, por isso 128 está bem na metade desse intervalo. Dessa forma, o ciclo de trabalho é de 50%, e os motores rodarão com metade da velocidade máxima. Independentemente da direção e velocidade escolhidas para seus motores, eles continuarão nesse sentido até que você as altere. Portanto, a espera de dois segundos serve para garantir que o robô avance a 50% de sua velocidade máxima por dois segundos, o que deve representar um metro, ou três pés.

Em seguida, você interrompe os motores e aguarda um segundo:

```
stopMotor(); delay(1000); // para
```

A próxima sequência de movimento inverte a direção do motor da esquerda, enquanto mantém o motor da direita para frente. Para fazer com que um motor avance, definimos seu pino de direção como `HIGH`; para fazer com que ele vá para trás, definimos como `LOW`. A velocidade permanece em 50%, por isso o valor de 128 é escrito para os pinos de velocidade de ambos os motores (as linhas de código que definem o movimento para frente, na roda da direita, e as velocidades não são estritamente necessárias, mas coloquei-as no código para que você possa modificá-las, se desejar obter movimentos diferentes).

Fazer com que a roda da direita avance e a da esquerda vá na direção oposta faz com que o robô rotacione em sentido anti-horário (vire para a esquerda). Esse movimento ocorre por um segundo, o que deve fazer com que o robô vire 90 graus para a esquerda. Depois dessa sequência, paramos o motor novamente por um segundo:

```
digitalWrite(direction1, LOW);
digitalWrite(direction2, HIGH);
analogWrite(speed1, 128);
analogWrite(speed2, 128);
delay(1000);
```

A próxima sequência faz com que o robô avance, mais uma vez, a 50% de sua velocidade máxima por dois segundos, seguido por uma parada de um segundo:

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, HIGH);
analogWrite(speed1,128);
analogWrite(speed2,128);
delay(2000);
```

A sequência final de movimento altera a direção das rodas, para que a roda da esquerda avance, enquanto a da direita vai para trás. Isso fará com que o robô vire no sentido horário (para a direita). A velocidade dessa vez é definida como 64, o que representa 25% do ciclo de trabalho, fazendo com que o robô rotacione mais lentamente do que antes. Essa rotação dura dois segundos, o que deve ser suficiente para girar o robô 90 graus.

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, LOW);
analogWrite(speed1, 64);
analogWrite(speed2, 64);
delay(2000);
```

As quatro sequências, quando tomadas juntas, devem fazer com que seu robô avance por dois segundos, pare, vire 90 graus para a esquerda, pare, avance novamente por dois segundos, pare, e então vire 90 graus para a direita, mais lentamente, e repita o processo desde o início.

Talvez seja necessário ajustar os tempos e as velocidades de acordo com seu próprio robô, uma vez que você pode estar utilizando uma voltagem diferente, motores engrenados mais lentos ou mais rápidos etc. Brinque um pouco com as sequências de movimento, fazendo com que o robô se mova mais rápido ou mais lentamente, ou vire graus diferentes de acordo com sua vontade.

O objetivo do projeto 29 é fazer com que você se acostume com os comandos necessários para movimentar dois motores em direções diferentes e utilizando velocidades diferentes. Se você tem apenas os motores, mas nenhuma base de robô, prenda um pedaço de fita ao eixo dos motores para que possa ver a velocidade e a direção em que eles estão girando.

Projeto 29 – Uso de um shield de motor – Análise do hardware

O novo componente de hardware que utilizamos neste projeto é um shield de motor. Um shield é simplesmente uma placa de circuito, construída e preparada para se encaixar devidamente ao Arduino. O shield tem pinos que permitem conectá-lo aos pinos do Arduino, com soquetes fêmeas no topo, para que você continue tendo acesso aos pinos passando pelo shield. Evidentemente, dependendo do tipo de shield que você utiliza, alguns dos pinos do Arduino serão utilizados pelo shield e não estarão disponíveis em seu código. O shield Ardumoto, por exemplo, utiliza os pinos digitais 10, 11, 12 e 13.

Shields têm o benefício de oferecer funcionalidades adicionais ao seu Arduino, bastando para isso que sejam conectados e que você faça o upload do código. Há todo tipo de shields, capazes de estender as funções do Arduino para incluir elementos como acesso Ethernet, GPS, controle de relé, acesso a cartões SD e Micro SD, displays LCD, conectividade com TVs, comunicação wireless, telas sensíveis ao toque, displays de matriz de pontos e controle de iluminação DMX. Veja a figura 10.8, para alguns exemplos de tipos diferentes de shields.

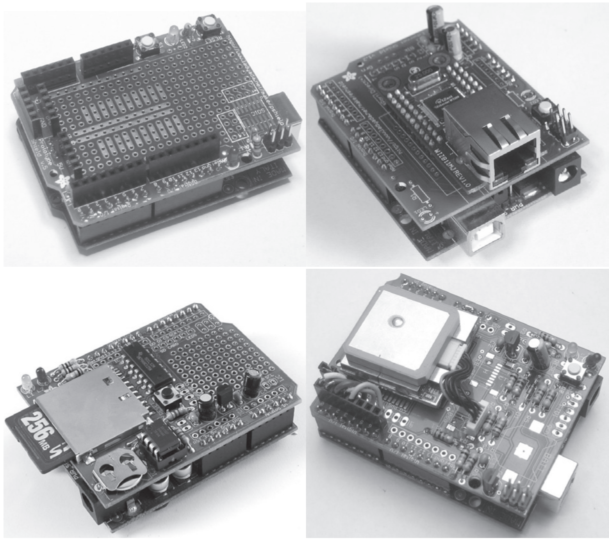


Figura 10.8 – Protoshield, shield para Ethernet, shield registrador de dados, e um shield GPS (cortesia da Ardafruit Industries, em www.adafruit.com).

Projeto 30 – Robô que acompanha uma linha

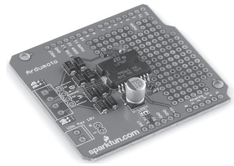
Agora que você sabe como controlar dois motores, ou sua base de robô, utilizando um shield, vamos colocar essas habilidades em prática. Neste projeto, você criará um robô capaz de detectar e acompanhar uma linha desenhada no chão. Esse tipo de robô ainda é utilizado em fábricas, para garantir que o robô se movimente apenas em um caminho específico, determinado no piso da fábrica. Eles são conhecidos como veículos guiados automaticamente (Automated Guided Vehicles, ou AGVs).

Componentes necessários

Para este projeto, você necessitará de uma base de robô pequena e leve. Bases desse tipo podem ser adquiridas prontas ou na forma de kits. Você mesmo também pode construir uma do zero (o que é muito mais divertido). Certifique-se de que as rodas estejam engrenadas, de modo a rotacionarem de forma muito mais lenta do que o motor, uma vez que o robô deve ser capaz de se movimentar lentamente, para que possa acompanhar a linha.

Como o robô será autônomo, necessitará de sua própria fonte de alimentação. Verifiquei que um suporte com seis pilhas AA forneceu energia suficiente, em meu caso, para o Arduino, os sensores, os LEDs, o shield e os motores. Pilhas maiores fornecerão uma vida mais longa ao seu robô, mas ao custo de serem mais pesadas, portanto, tudo dependerá da capacidade de carga de seu robô.

Shield de motor



4 resistores limitadores de corrente



3 resistores de 1 kΩ



4 LEDs brancos



3 resistores dependentes de luz



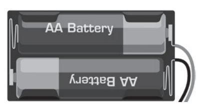
2 motores CC ou...



... uma base de robô de duas rodas



Fonte de alimentação



Conectando os componentes

Conecte o circuito como na figura 10.9. O shield não é mostrado, mas as quatro saídas simplesmente vão para o Motor A e o Motor B.

Os quatro LEDs podem ser de qualquer cor, desde que tenham brilho suficiente para iluminar o solo e criar um contraste, entre o piso e a linha preta, que seu sensor possa detectar. LEDs brancos são melhores, uma vez que emitem o maior brilho, com a menor corrente. Quatro LEDs desse tipo são conectados em paralelo com o pino 9. Certifique-se de que você esteja utilizando os resistores limitadores de corrente apropriados para seus LEDs.

Por fim, conecte três resistores limitadores de corrente aos pinos analógicos 0, 1 e 2. Um lado do LDR (resistor dependente de luz) vai para o terra, e o outro para os +5 V, via um resistor de 1 kΩ. A fiação dos pinos analógicos passa entre o resistor e o pino do LDR (você utilizou LDRs no projeto 14).

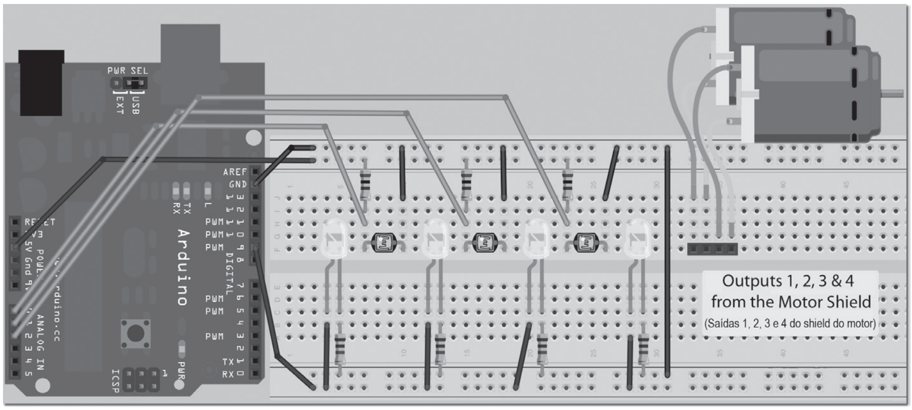


Figura 10.9 – Circuito para o Projeto 30 – Robô que acompanha uma linha (consulte o site da Novatec para versão colorida).

A protoboard que temos na figura 10.9 pode ser utilizada para testes. Entretanto, para criar o robô, você terá de criar uma barra sensora (*sensor bar*) utilizando os quatro LEDs e os três LDRs. Estes devem estar posicionados de uma forma que estejam todos próximos e apontando para a mesma direção. Soldei o LED e o circuito do sensor em uma pequena tira de stripboard, colocando os três LDRs entre os quatro LEDs e apenas um pouco mais alto que eles, para evitar que a luz dos LEDs afetasse o sensor. A figura 10.10 mostra como fiz essa instalação.

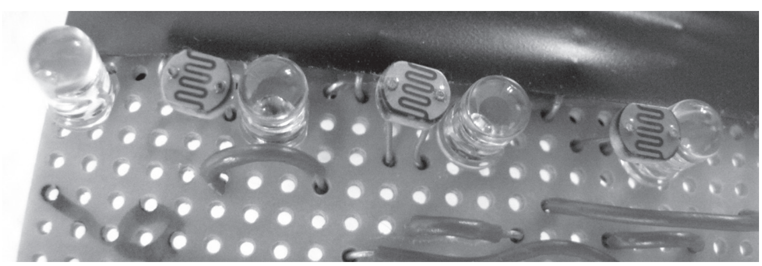


Figura 10.10 – Barra sensora consistindo de quatro LEDs e três LDRs.

A barra sensora é presa à frente de seu robô, de modo que os LEDs e os LDRs fiquem aproximadamente a um centímetro, ou meia polegada, do solo (Figura 10.11). Simplesmente prendi minha barra sensora com uma fita à frente do meu robô, mas para uma solução mais permanente, você pode parafusar a barra sensora ao chassi.

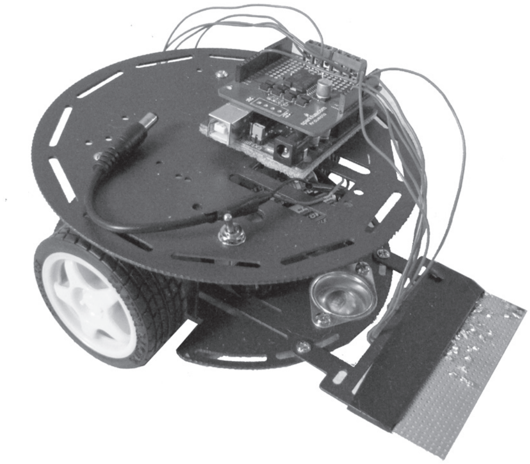


Figura 10.11 – Base de robô com uma barra sensora.

A fonte de alimentação deve ser posicionada centralmente, dentro da base do robô, para que não desequilibre as rodas. Prenda as pilhas ao chassi com velcro, para que elas não se movimentem conforme o robô percorre seu caminho.

Como você pode ver na figura 10.11, fios se estendem entre o shield do motor e os dois motores, assim como por baixo da barra sensora. Um fio fornece os +5 V, outro é para o terra, um terceiro para os +5 V do pino digital 9 e outros três para os sensores analógicos 0, 1 e 2. 0 é o sensor da esquerda (quando olhamos o robô de frente), 1 é o sensor do centro, e 2, o sensor da direita. É vital que você utilize a ordem correta, ou o robô não funcionará como esperado. Veja a figura 10.12 para uma imagem de meu robô em ação, no chão de minha cozinha (há também um vídeo dele funcionando no YouTube e no Vimeo, se você conseguir encontrá-lo).

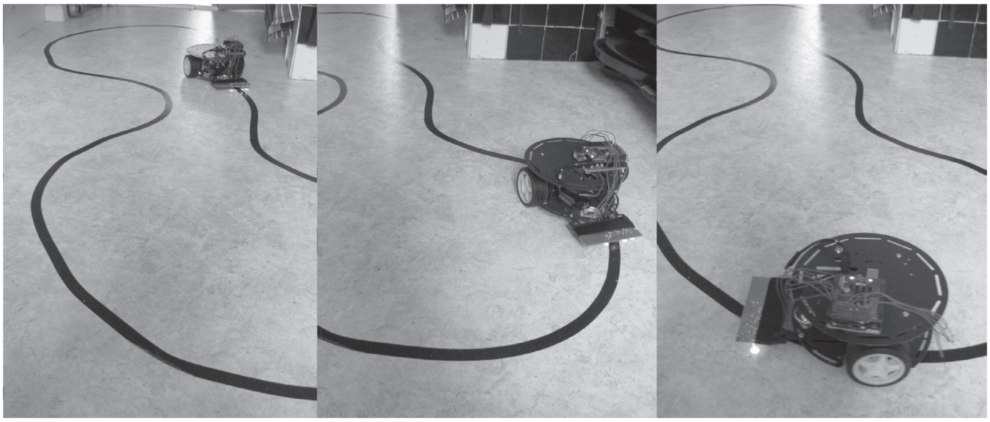


Figura 10.12 – Meu próprio robô que acompanha uma linha.

Digite o código

Depois de ter construído seu robô e verificado todas as conexões, digite o código da listagem 10.3.

Listagem 10.3 – Código para o projeto 30

```
// Projeto 30 - Robô que acompanha uma linha

#define lights 9
int LDR1, LDR2, LDR3;    // valores dos sensores

// deslocamentos (offsets) de calibração
int leftOffset = 0, rightOffset = 0, centre = 0;
// pinos para a velocidade e direção do motor
int speed1 = 3, speed2 = 11, direction1 = 12, direction2 = 13;
// velocidade inicial e deslocamento da rotação
int startSpeed = 70, rotate = 30;
// limiar do sensor
int threshold = 5;
// velocidades iniciais dos motores esquerdo e direito
int left = startSpeed, right = startSpeed;

// Rotina de calibração do sensor
void calibrate() {

  for (int x=0; x<10; x++) {   // executa 10 vezes, para obter uma média
  digitalWrite(lights, HIGH);  // acende as luzes
  delay(100);
  LDR1 = analogRead(0);      // lê os 3 sensores
  LDR2 = analogRead(1);
  LDR3 = analogRead(2);
  leftOffset = leftOffset + LDR1;   // adiciona o valor do sensor da esquerda ao total
  centre = centre + LDR2;           // adiciona o valor do sensor do centro ao total
  rightOffset = rightOffset + LDR3;   // adiciona o valor do sensor da direita ao total

  delay(100);
  digitalWrite(lights, LOW);   // apaga as luzes
  delay(100);
  }
  // obtém a média para cada sensor
  leftOffset = leftOffset / 10;
  rightOffset = rightOffset / 10;
  centre = centre / 10;
  // calcula os deslocamentos para os sensores esquerdo e direito
  leftOffset = centre - leftOffset;
```

```
  rightOffset = centre - rightOffset;
}

void setup() {
  // define os pinos dos motores como saídas
  pinMode(lights, OUTPUT);   // luzes
  pinMode(speed1, OUTPUT);
  pinMode(speed2, OUTPUT);
  pinMode(direction1, OUTPUT);
  pinMode(direction2, OUTPUT);
  // calibra os sensores
  calibrate();
  delay(3000);

  digitalWrite(lights, HIGH);    // acende as luzes
  delay(100);

  // define a direção do motor para frente
  digitalWrite(direction1, HIGH);
  digitalWrite(direction2, HIGH);
  // define a velocidade de ambos os motores
  analogWrite(speed1,left);
  analogWrite(speed2,right);
}

void loop() {
  // utiliza a mesma velocidade em ambos os motores
  left = startSpeed;
  right = startSpeed;

  // lê os sensores e adiciona os deslocamentos
  LDR1 = analogRead(0) + leftOffset;
  LDR2 = analogRead(1);
  LDR3 = analogRead(2) + rightOffset;
  // se LDR1 for maior do que o sensor do centro + limiar, vire para a direita
  if (LDR1 > (LDR2+threshold)) {
    left = startSpeed + rotate;
    right = startSpeed - rotate;
  }
  // se LDR3 for maior do que o sensor do centro + limiar, vire para a esquerda
  if (LDR3 > (LDR2+threshold)) {
    left = startSpeed - rotate;
    right = startSpeed + rotate;
  }
```

```
    // envia os valores de velocidade para os motores
    analogWrite(speed1,left);
    analogWrite(speed2,right);
}
```

Elabore um percurso para seu robô em uma superfície plana. No meu caso, utilizei o linóleo do chão de minha cozinha e criei um percurso utilizando fita isolante preta. Ligue o robô, certificando-se de que ele esteja posicionado sobre uma área vazia do piso, próxima, mas não sobre a linha preta. A rotina de calibração será executada, fazendo com que os LEDs pisquem dez vezes em rápida sucessão. Assim que isso terminar, você terá três segundos para pegar o robô e colocá-lo sobre a linha. Se tudo der certo, seu robô acompanhará o trajeto. Ao criar o percurso, não faça curvas muito fechadas, uma vez que essa rápida transição não será detectada com apenas três LDRs. Veja a figura 10.12, para um exemplo de um percurso desenhado com fita isolante preta.

Projeto 30 – Robô que acompanha uma linha – Análise do código

Primeiramente, você define o pino para as luzes, e declara três inteiros para armazenar os valores lidos dos três resistores dependentes de luz:

```
#define lights 9
int LDR1, LDR2, LDR3;  // valores dos sensores
```

Depois, outros três inteiros são declarados para armazenar os valores de deslocamento (*offset*) dos três sensores, calculados na rotina de calibração (isso será explicado mais adiante):

```
int leftOffset = 0, rightOffset = 0, centre = 0;
```

Na sequência, você define os pinos que controlarão a velocidade e a direção dos dois motores:

```
int speed1 = 3, speed2 = 11, direction1 = 12, direction2 = 13;
```

Então, dois inteiros são criados para armazenar a velocidade inicial dos motores e a diferença de velocidade para cada roda, que você adicionará ou subtrairá para que o robô rotacione:

```
int startSpeed = 70, rotate = 30;
```

A velocidade padrão é definida como `70`, praticamente 27% do ciclo de trabalho. Pode ser necessário ajustar esse valor de acordo com seu próprio robô. Um valor alto demais fará com que o robô ultrapasse a linha, enquanto um baixo demais impedirá que os motores girem rápido o bastante para movimentar as rodas e superar a fricção.

O valor de `rotate` indica quanto você acelerará ou desacelerará as rodas, para fazer com que o robô vire. Em meu caso, o valor necessário é `30`. Assim, quando o robô

vira para a esquerda, por exemplo, a roda da direita gira à velocidade 100 e a roda da esquerda à velocidade 40 (70 + 30 e 70 - 30). O valor de `rotate` é outra definição que você pode ter de ajustar em sua configuração.

Mais um inteiro é declarado para armazenar o limiar (*threshold*) do sensor:

```
int threshold = 5;
```

O limiar é a diferença de valor necessária entre o sensor do centro e os sensores da direita e da esquerda, para que o robô decida se deve virar. Em meu caso, uma definição com valor `5` é suficiente. Isso significa que os sensores da direita e da esquerda terão de detectar um valor maior do que o lido no sensor do centro, mais o valor do limiar, antes que tomem uma ação. Em outras palavras, se o sensor do centro estiver lendo um valor de `600`, e o sensor da esquerda, de `603`, o robô continuará andando em linha reta. Entretanto, um valor no sensor da esquerda de `612` (maior que o valor do centro mais o limiar) significará que o sensor da esquerda não está detectando a linha preta, indicando que o robô está muito para a esquerda. Nesse caso, os motores seriam ajustados para virar o robô para a direita e compensar.

O valor de `threshold` variará dependendo do contraste entre seu piso (ou a superfície utilizada) e a linha preta. Ele pode ter de ser ajustado, para garantir que seu robô vire apenas quando for detectada uma diferença suficiente entre o piso e a linha, para garantir que ele não avance demais para a esquerda ou para a direita.

O conjunto final de variáveis armazenará os valores de velocidade para os motores da esquerda e da direita. Inicialmente, eles são definidos como o valor em `startSpeed`:

```
int left = startSpeed, right = startSpeed;
```

Depois das variáveis terem sido declaradas e inicializadas, você chega à sua primeira e única função, a rotina de calibração:

```
void calibrate() {
```

O propósito dessa rotina é duplo. Primeiramente, ela obtém um valor médio de cada um três sensores em uma sequência de dez leituras. Depois, pisca as luzes dez vezes (enquanto lê os valores dos três sensores) como uma forma de mostrar a você que o robô está calibrando e quase pronto para ser utilizado.

Os sensores requerem calibração, uma vez que cada um lerá valores diferentes dos demais. Cada LDR fará uma leitura levemente diferente, afetada por valores de tolerância do fabricante, pela tolerância do resistor divisor de tensão utilizado, e pela resistência dos fios. Você deseja que todos os três sensores leiam (aproximadamente) o mesmo valor, por isso realiza dez leituras em cada um, determina uma média e calcula a diferença entre os sensores da esquerda e da direita e o sensor do centro (utilizado como linha-base).

Você inicia criando um loop `for`, a ser executado dez vezes:

```
for (int x=0; x<10; x++) {
```

Os LEDs anexados ao pino 9 são acesos, seguidos por uma espera de cem milissegundos:

```
digitalWrite(9, HIGH);
delay(100);
```

Depois, os valores de todos os três sensores são lidos e armazenados em `LDR1`, `LDR2` e `LDR3`:

```
LDR1 = analogRead(0);
LDR2 = analogRead(1);
LDR3 = analogRead(2);
```

Agora você pega esses valores, adicionando-os às variáveis `leftOffset`, `center` e `rightOffset`. Essas variáveis iniciam em zero, por isso, depois de dez iterações, elas terão um total de todos os dez valores lidos dos sensores.

```
leftOffset = leftOffset + LDR1;
centre = centre + LDR2;
rightOffset = rightOffset + LDR3;
```

Em seguida, você espera cem milissegundos, apaga a luz, espera mais cem milissegundos e, então, repete o processo:

```
delay(100);
digitalWrite(9, LOW);    // apaga as luzes
delay(100);
```

Depois de repetir esse processo dez vezes, você sai do loop `for` e divide cada um dos totais por `10`. Isso resulta em uma leitura média do sensor para cada um dos três LDRs.

```
leftOffset = leftOffset / 10;
rightOffset = rightOffset / 10;
centre = centre / 10;
```

Então, você calcula qual será o deslocamento, subtraindo dos valores dos sensores da esquerda e da direita o valor do sensor do centro:

```
leftOffset = centre - leftOffset;
rightOffset = centre - rightOffset;
```

Esses valores serão adicionados às leituras dos sensores da esquerda e da direita, de modo que todos os três sensores tenham aproximadamente as mesmas leituras. Isso fará com que a verificação da diferença entre as três leituras seja muito mais fácil, quando estivermos detectando a diferença entre o piso e a linha preta.

Em seguida, você tem a rotina de inicialização, que inicia definindo o pino para os LEDs e os pinos para a velocidade e direção do motor como `OUTPUT`:

```
pinMode(9, OUTPUT);    // luzes
pinMode(speed1, OUTPUT);
pinMode(speed2, OUTPUT);
pinMode(direction1, OUTPUT);
pinMode(direction2, OUTPUT);
```

A rotina de calibração é executada, para garantir que todos os três sensores tenham leituras semelhantes, seguida por uma espera de três segundos. Depois de os LEDs piscarem dez vezes durante a rotina de calibração, você terá três segundos para posicionar o robô sobre a linha que ele deverá seguir:

```
calibrate();
delay(3000);
```

As luzes são acesas, seguidas por uma breve espera:

```
digitalWrite(9, HIGH);   // acende as luzes
delay(100);
```

Definimos a direção de ambos os motores para frente, e as velocidades com os valores armazenados nas variáveis `right` e `left` (inicialmente definidas com o valor armazenado em `startSpeed`):

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, HIGH);
analogWrite(speed1,left);
analogWrite(speed2,right);
```

Agora você avança para o loop principal do programa, que inicia definindo a velocidade dos motores da esquerda e da direita com o valor em `startSpeed`:

```
left = startSpeed;
right = startSpeed;
```

A velocidade do motor é redefinida com esses valores no início de cada loop, de modo que o robô esteja sempre indo para frente, a menos que os valores sejam alterados posteriormente no loop, fazendo com que o robô vire.

Agora, você lê os valores do sensor de cada um dos três LDRs e armazena o resultado nos inteiros `LDR1`, `LDR2` e `LDR3`. Os deslocamentos calculados para os sensores da esquerda e da direita são adicionados aos valores lidos, para que, quando os três sensores estiverem analisando a mesma superfície, tenham aproximadamente os mesmos valores.

```
LDR1 = analogRead(0) + leftOffset;
LDR2 = analogRead(1);
LDR3 = analogRead(2) + rightOffset;
```

Agora você tem de verificar os valores dos sensores e descobrir se a linha preta se afastou demais do centro. Isso é feito verificando os sensores da esquerda e da direita

e analisando se os valores lidos são maiores do que o valor lido no LDR do centro, somado ao deslocamento do limiar.

Se o valor do sensor da esquerda for maior do que a leitura do centro mais o deslocamento, então a linha preta terá se afastado do centro e estará muito à direita. As velocidades dos motores serão ajustadas para que a roda da esquerda gire mais rápido do que a da direita, virando o robô para a direita e retornando a linha preta ao centro:

```
if (LDR1 > (LDR2+threshold)) {
  left = startSpeed + rotate;
  right = startSpeed - rotate;
}
```

O mesmo é feito para o sensor da direita, dessa vez virando o robô para a esquerda:

```
if (LDR3 > (LDR2+threshold)) {
  left = startSpeed - rotate;
  right = startSpeed + rotate;
}
```

Então, as velocidades, que podem ou não ter sido ajustadas dependendo da posição da linha, são enviadas aos motores:

```
analogWrite(speed1,left);
analogWrite(speed2,right);
```

Todo esse processo se repete muitas vezes por segundo, e forma um loop de feedback que faz com que o robô siga a linha.

Caso seu robô saia do trajeto da linha, talvez você tenha de ajustar a velocidade em `startSpeed` para uma mais lenta. Se ele não virar com a rapidez ou extensão necessária, ou se virar demais, então o valor de `rotate` terá de ser ajustado de acordo. A sensibilidade dos LDRs pode ser ajustada, alterando o valor na variável `threshold`. Experimente valores diferentes, até que você consiga que o robô siga a linha devidamente.

Resumo

O capítulo 10 iniciou com uma simples introdução aos motores de passo, e explicou como controlar sua velocidade e direção utilizando um CI controlador de motor. Você também aprendeu qual a diferença entre um motor de passo unipolar e um bipolar, e como eles funcionam. Depois, você utilizou um simples shield de motor para controlar dois motores de forma independente, e terminou conectando esse shield a uma base de robô, para criar um robô de duas rodas que acompanha uma linha desenhada no chão.

Você também aprendeu o que são shields e suas aplicações diferentes, além de ter utilizado, na prática, um shield de motor. Sabendo como controlar motores de passo e motores normais, assim como servos (Capítulo 9), você está bem encaminhado para criar seu próprio robô avançado, ou algum dispositivo de animatronics.

Assuntos e conceitos abordados no capítulo 10:

- a diferença entre motores de passo unipolares e bipolares;
- como conectar um motor de passo a um CI controlador de motor;
- como utilizar capacitores para suavizar um sinal e reduzir interferência;
- como configurar um objeto `stepper` em seu código;
- como definir a velocidade e o número de passos do motor;
- como controlar a direção do motor com números positivos e negativos;
- como funciona um motor de passo;
- a sequência de energização da bobina necessária para controlar um motor de passo;
- como elevar a resolução utilizando os recursos de meio-passo e micropasso;
- como utilizar um shield de motor;
- como utilizar um shield de motor para controlar a velocidade e/ou direção de vários motores;
- o que é um shield do Arduino e os diferentes tipos de shields disponíveis;
- como detectar variações de contraste de luz utilizando diversos LDRs;
- como obter uma leitura média de um sensor;
- como utilizar valores de deslocamento para balancear leituras de sensores;
- como criar um incrível robô que acompanha uma linha desenhada no chão.

CAPÍTULO 11

Sensores de pressão

Agora você trabalhará com sensores de pressão, em especial com o sensor digital de pressão absoluta SCP1000, da VTI, um ótimo sensor, cuja interface com o Arduino é simples e que fornece leituras precisas de pressão e temperatura. O dispositivo tem de ser conectado ao barramento da SPI (Serial Peripheral Interface, ou interface periférica serial) do Arduino, para que os dados sejam transmitidos. A SPI é um conceito novo para você, e ainda que este capítulo não pretenda abordá-la em grandes detalhes (seria necessário um ou dois capítulos apenas para tanto), você aprenderá os conceitos básicos desse tópico, e como utilizá-los para obter dados do sensor SCP1000.

Sensores de pressão absoluta são ideais para criação de sua própria estação meteorológica. Você iniciará este capítulo aprendendo como fazer a interface do SCP1000 com o Arduino e como ler seus dados utilizando o monitor serial. Em seguida, você utilizará um sensor e um LCD para exibir um gráfico de pressão em um período de 24 horas. Nesse processo, você aprenderá como controlar um display GLCD (Graphic LCD).

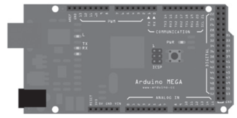
Projeto 31 – Sensor digital de pressão

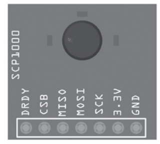
Você utilizará um Arduino Mega para este projeto, uma vez que o projeto 32 utilizará um GLCD e os pinos adicionais do Mega serão necessários. Se você não tem um Mega, pode utilizar seu Arduino-padrão — basta alterar os pinos da SPI para corresponder àqueles de um Duemilanove.

Componentes necessários

O pequenino sensor SCP1000 pode ser adquirido na Sparkfun, ou em seus distribuidores, pré-soldado a uma placa breakout[1] (Figura 11.1), para facilitar a interface com um Arduino (ou outro microcontrolador). Você terá de soldar alguns pinos de cabeçalho à placa, caso queira conectá-la a uma protoboard. Do contrário, solde alguns fios a ela, para que possa conectá-la ao Mega.

1 N.T.: O Capítulo 12 falará sobre placas breakout com mais detalhes.

Arduino Mega 

Sensor de pressão SCP1000 

3 resistores de 10 kΩ 

1 resistor de 1 kΩ 

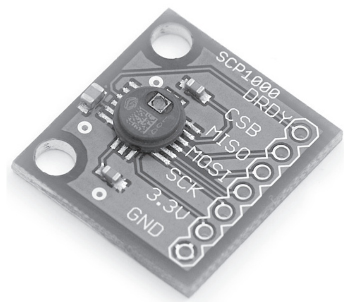


Figura 11.1 – SCP1000 em placa breakout da Sparkfun.

Conectando os componentes

Conecte tudo como mostra a figura 11.2.

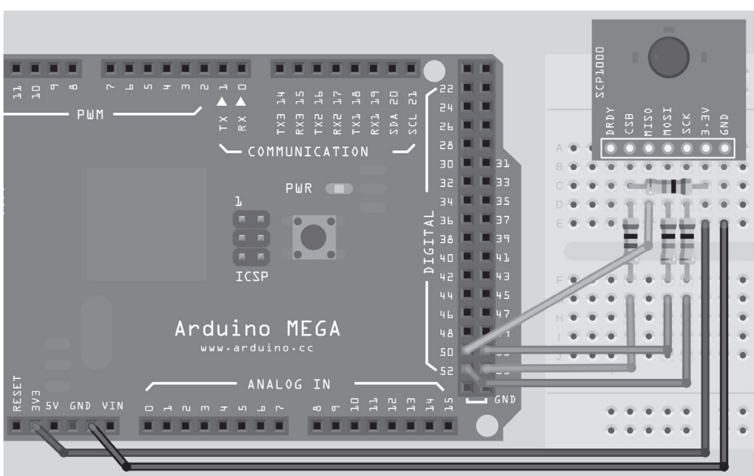


Figura 11.2 – Circuito para o Projeto 31 – Sensor digital de pressão (consulte o site da Novatec para versão colorida).

O sensor tem pinos TRIG e PD que também terão de ser conectados ao terra. Certifique-se de que tudo esteja conectado corretamente, especialmente os resistores, para proteger o SCP1000 contra sobretensão. Agora, digite o código.

Digite o código

Digite o código da listagem 11.1.

Listagem 11.1 – Código para o projeto 31

```
/*
SCP1000    Mega
DRDY       N/D
CSB        53 via conversor de nível lógico
MISO       50 (direto)
MOSI       51 via conversor de nível lógico
SCK        52 via conversor de nível lógico
3.3 V      3.3 V
GND        Terra (GND)
TRIG       Terra (GND)
PD         Terra (GND)
*/

// Pinos da SPI
#define SLAVESELECT 53
#define SPICLOCK 52
#define DATAOUT 51   // MOSI
#define DATAIN 50  // MISO
#define UBLB(a,b)  ( ( (a) << 8) | (b) )
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )

// Endereços
#define PRESSURE 0x1F       // 3 bits mais significativos da pressão
#define PRESSURE_LSB 0x20   // 16 bits menos significativos da pressão
#define TEMP 0x21           // 16 bits da temperatura

char rev_in_byte;
int temp_in;
unsigned long pressure_lsb;
unsigned long pressure_msb;
unsigned long temp_pressure;
unsigned long pressure;

void setup() {
  byte clr;
  pinMode(DATAOUT, OUTPUT);
```

```
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK, OUTPUT);
  pinMode(SLAVESELECT, OUTPUT);
  digitalWrite(SLAVESELECT, HIGH);    // desativa o dispositivo

  SPCR = B01010011;    // SPI Control Register (Registrador de controle da SPI)
  // MPIE=0, SPE=1 (on), DORD=0 (MSB primeiro), MSTR=1 (master), CPOL=0 (clock ocioso quando low),
  // CPHA=0 (faz a amostragem do MOSI durante a extremidade ascendente), SPR1=0 & SPR0=0 (500 kHz)
  clr=SPSR;  // SPI Status Register (Registrador de estado da SPI)
  clr=SPDR;  // SPI Data Register (Registrador de dados da SPI)
  delay(10);
  Serial.begin(38400);
  delay(500);

  write_register(0x03,0x09);   // Modo de leitura de alta velocidade
  write_register(0x03,0x0A);   // Modo de medição de alta resolução
}

void loop() {
  pressure_msb = read_register(PRESSURE);
  pressure_msb &= B00000111;
  pressure_lsb = read_register16(PRESSURE_LSB);
  pressure_lsb &= 0x0000FFFF;
  pressure = UBLB19(pressure_msb, pressure_lsb);
  pressure /= 4;

  Serial.print("Pressure (hPa): ");
  float hPa = float(pressure)/100;
  Serial.println(hPa);

  Serial.print("Pressure (Atm): ");
  float pAtm = float(pressure)/101325.0;
  Serial.println(pAtm, 3);

  temp_in = read_register16(TEMP);
  float tempC = float(temp_in)/20.0;
  Serial.print("Temp. C: ");
  Serial.println(tempC);
  float tempF = (tempC*1.8) + 32;
  Serial.print("Temp. F: ");
  Serial.println(tempF);
  Serial.println();
  delay(1000);
}
```

```
char spi_transfer(char data) {
  SPDR = data;              // Inicia a transmissão
  while (!(SPSR & (1<<SPIF))) { };    // Aguarda o fim da transmissão
  return SPDR;              // retorna o byte recebido
}

char read_register(char register_name) {
  char in_byte;
  register_name <<= 2;
  register_name &= B11111100;    // Comando de leitura

  digitalWrite(SLAVESELECT, LOW);// Ativa o dispositivo SPI
  spi_transfer(register_name);   // Escreve o byte para o dispositivo
  in_byte = spi_transfer(0x00);    // Não envia nada, mas pega de volta o valor do registro
  digitalWrite(SLAVESELECT, HIGH);  // Desativa o dispositivo SPI
  delay(10);
  return(in_byte);  // valor de retorno
}

unsigned long read_register16(char register_name) {
  byte in_byte1;
  byte in_byte2;
  float in_word;

  register_name <<= 2;
  register_name &= B11111100;    // Comando de leitura

  digitalWrite(SLAVESELECT, LOW);   // Ativa o dispositivo
  spi_transfer(register_name);      // Escreve o byte para o dispositivo
  in_byte1 = spi_transfer(0x00);
  in_byte2 = spi_transfer(0x00);
  digitalWrite(SLAVESELECT, HIGH);    // Desativa o dispositivo
  in_word = UBLB(in_byte1,in_byte2);
  return(in_word);          // valor de retorno
}

void write_register(char register_name, char register_value) {
  register_name <<= 2;
  register_name |= B00000010;    // Comando de escrita

  digitalWrite(SLAVESELECT, LOW); // Seleciona o dispositivo SPI
  spi_transfer(register_name);   // Envia a localização do registrador
  spi_transfer(register_value);    // Envia o valor a ser gravado no registrador
  digitalWrite(SLAVESELECT, HIGH);
}
```

Esse código tem por base o trabalho de Conor e de outros colaboradores dos fóruns do Arduino, por isso agradecemos a todos os envolvidos. Além disso, a versão 0019 do IDE do Arduino veio com um sketch de exemplo da SPI para o SCP1000, utilizando a biblioteca `SPI.h`, de Tom Igoe.

O código deste projeto executa as operações do modo mais difícil, para que você possa entender exatamente como o SCP1000 se comunica com o Arduino. Assim que você compreender esses dados, poderá facilitar seu trabalho utilizando a biblioteca SPI.

Depois do upload do código, abra a janela do monitor serial e verifique se sua taxa de transmissão está definida como 38400. Você verá um fluxo de dados do sensor mostrando a pressão em hPa (hectopascals), assim como em atmosferas. Hectopascal é a unidade de pressão mais usual em previsões meteorológicas. Você também verá a temperatura em Celsius e Fahrenheit.

Projeto 31 – Sensor digital de pressão – Análise do código

O código inicia com um conjunto de `#defines` para seus pinos no Mega:

```
#define SLAVESELECT 53
#define SPICLOCK 52
#define DATAOUT 51   // MOSI
#define DATAIN 50    // MISO
```

Se estiver utilizando um Duemilanove, os pinos serão:

```
#define SLAVESELECT 10
#define SPICLOCK 13
#define DATAOUT 11   // MOSI
#define DATAIN 12  // MISO
```

Em seguida, temos outros dois `#defines`, inteligentemente projetados para realizar operações de deslocamento de bits:

```
#define UBLB(a,b)  ( ( (a) << 8) | (b) )
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )
```

A primeira diretiva converterá dois dígitos de 8 bits em um dígito de 16 bits, utilizando um deles como o LSB (byte menos significativo) e outro como o MSB (byte mais significativo). Isso é feito utilizando deslocamento de bits e algumas operações bit a bit.

Por exemplo, se você tivesse dois números de 8 bits (10010101 e 00111001) e aplicasse neles a seguinte equação:

```
UBLB(a,b)  ( ( (a) << 8) | (b) )
```

O cálculo seria:

```
(((B10010101) << 8) | (B00111001))
```

Assim, o primeiro dígito seria deslocado para a esquerda oito vezes, criando:

```
B1001010100000000
```

A esse número aplicamos um OU bit a bit com o segundo dígito, para criar:

```
B1001010100111001
```

Com isso, simplesmente pegamos dois números de 8 bits e os convertemos em um número de 16 bits, utilizando um conjunto de 8 bits como o MSB e outro como o LSB.

O segundo `#define` faz algo semelhante:

```
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )
```

Dessa vez, ele criará um número de 19 bits, a resolução do sensor de pressão SCP1000. Isso é feito deslocando os primeiros três bits 16 posições para a esquerda, deixando 16 bits disponíveis para adicionar o dígito de 16 bits armazenado em `b`.

Na sequência, você define os três registros dentro do sensor, que devem ser lidos para obter os dados de pressão e temperatura:

```
#define PRESSURE 0x1F       // 3 bits mais significativos da pressão
#define PRESSURE_LSB 0x20   // 16 bits menos significativos da pressão
#define TEMP 0x21           // 16 bits da temperatura
```

O registro `PRESSURE`, no endereço `0x1F`, armazena os três bits mais significativos do número de 19 dígitos que compõe a leitura de pressão. O registro `PRESSURE_LSB`, no endereço `0x20`, armazena os 16 bits seguintes do número. Utilizando o cálculo definido em `UBLB19(a,b)`, combinamos os números de 16 bits e de 3 bits, para criar um número de 19 bits.

Em seguida, você declara algumas variáveis utilizadas para armazenar os valores lidos dos sensores de pressão e temperatura:

```
char rev_in_byte;
int temp_in;
unsigned long pressure_lsb;
unsigned long pressure_msb;
unsigned long temp_pressure;
unsigned long pressure;
```

Na sequência, temos a rotina de inicialização. Você começa declarando uma variável local de tipo `byte` (veremos como ela é utilizada em breve):

```
byte clr;
```

Depois, você define os quatro pinos que formam o barramento da SPI, com seus respectivos estados `INPUT` e `OUTPUT`:

```
pinMode(DATAOUT, OUTPUT);
pinMode(DATAIN, INPUT);
```

```
pinMode(SPICLOCK, OUTPUT);
pinMode(SLAVESELECT, OUTPUT);
```

Então, você define o pino Slave Select (Seleção de Escravo) como `HIGH` para desabilitar o dispositivo, uma vez que você não deseja trocar dados com o dispositivo, enquanto ainda efetua sua configuração:

```
digitalWrite(SLAVESELECT, HIGH);  // desativa o dispositivo
```

Em seguida, você define `SPCR` com o valor binário de B01010011:

```
SPCR = B01010011;   // SPI Control Register (Registrador de controle da SPI)
```

Você pode ter notado que essa variável ainda não foi declarada, mas o código funcionará mesmo assim. Como isso é possível? Bem, SPCR significa SPI Control Register (Registrador de controle da SPI). É um dos três registradores utilizados em comunicações SPI, codificados diretamente no IDE do Arduino. Os outros dois são SPSR (SPI Status Register, ou registrador de estado da SPI) e SPDR (SPI Data Register, ou registrador de dados da SPI). Sempre que você atribuir valores a qualquer um desses registradores, alterará o que está acontecendo com o barramento da SPI.

Façamos uma pequena pausa para analisar a SPI e seu funcionamento.

SPI – Interface periférica serial

Antes que você continue a analisar o código, é necessário que compreenda o que é a SPI e como ela funciona. Como este pode ser um tópico complicado, não o abordaremos em muita profundidade. Afinal, este é um livro para iniciantes. Em vez disso, mostrarei o suficiente para que você compreenda o que é a SPI, como ela funciona e suas partes mais relevantes para os sketches dos projetos 31 e 32, que utilizam o barramento da SPI com o sensor SCP1000. Dessa forma, você será capaz de compreender como é feita a interface com outros dispositivos que também utilizam a SPI.

A SPI é uma forma de troca de informação entre dois dispositivos, e tem como benefício o fato de que exige apenas quatro pinos do Arduino, além de ser rápida. A SPI é um protocolo síncrono que permite a um dispositivo mestre (master) se comunicar com um dispositivo escravo (slave). Os dados são controlados com um sinal de clock (CLK), que decide quando eles podem ser trocados e quando são válidos para leitura. A taxa do clock pode variar, diferentemente de alguns outros protocolos em que o sinal do clock tem de ser cronometrado com muita precisão. Isso faz com que a SPI seja ideal para microcontroladores que não executam a grandes velocidades, ou nos quais o oscilador interno não é cronometrado precisamente.

A SPI é um protocolo Mestre-Escravo (Figura 11.3), o que significa que um dispositivo mestre controla o sinal do clock. Nenhum dado pode ser transmitido, a menos que haja um pulso do clock. A SPI também é um protocolo de troca de dados. Isso significa

que, conforme registramos a saída de dados, também ocorre a entrada de novos dados. Dispositivos não podem simplesmente transmitir ou receber dados; é necessário que haja uma troca, mesmo que um dos lados não esteja sendo utilizado em seu programa.

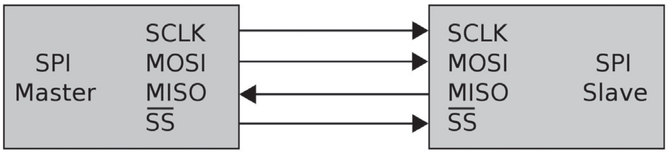


Figura 11.3 – Barramento da SPI: mestre e escravo únicos (imagem cortesia de Colin M. L. Burnett).

O pino Slave Select (SS) controla quando um dispositivo pode ser acessado, caso mais de um escravo esteja anexado ao mestre (Figura 11.4). Quando há apenas um dispositivo escravo, como em seu caso, o SS (CSB no SCP1000) é opcional. Entretanto, como regra, ele deve ser utilizado independentemente do caso, uma vez que também pode ser usado como um reset para o escravo, servindo para deixá-lo pronto a receber o próximo byte. O sinal de seleção de escravo é enviado pelo mestre, para dizer ao escravo que ele deseja iniciar uma troca de dados SPI. O sinal está ativo quando `LOW`, por isso, quando mantido `HIGH`, o dispositivo escravo não está selecionado.

A saída de dados ocorre apenas durante a extremidade ascendente ou descendente do sinal do clock no SCK (Serial Clock). Dados são "travados" (*latched*) durante a extremidade oposta do SCK. A polaridade do clock é definida pelo mestre, utilizando uma das flags definidas no registro SPCR.

As duas linhas de dados são conhecidas como MOSI (Master Output Slave Input, ou Saída-Mestre Entrada-Escravo) e MISO (Master Input Slave Output, Entrada-Mestre Saída-Escravo). Assim, se o dispositivo estiver definido para enviar dados do mestre na extremidade ascendente do pulso do clock, os dados serão enviados de volta do escravo na extremidade descendente. Dados, nesse caso, são enviados, tanto a partir do mestre (MOSI) quanto para o mestre (MISO), durante um pulso do clock.

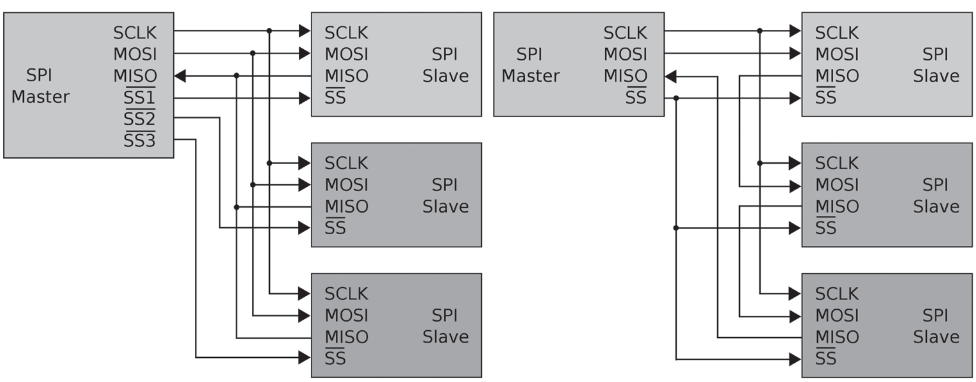


Figura 11.4 – Esquerda: um mestre com três escravos independentes. Direita: escravos encadeados (imagem cortesia de Colin M. L. Burnett).

Lembre-se de que, mesmo que você esteja interessado apenas em ler dados de um dispositivo (como está fazendo com o SCP1000), ainda terá de enviar dados nos dois sentidos, durante uma troca.

Os três registros utilizados pelo barramento da SPI são:

- SPCR – SPI Control Register (Registrador de controle da SPI);
- SPDR – SPI Data Register (Registrador de dados da SPI);
- SPSR – SPI Status Register (Registrador de estado da SPI).

O registrador de controle tem oito bits, e cada bit controla uma configuração específica da SPI. Os bits estão listados na tabela 11.1.

Tabela 11.1 – Definições do Registrador de Controle da SPI

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

- SPIE – SPI Interrupt Enable – Ativa a interrupção da SPI, se seu valor for 1;
- SPE – SPI Enable – Quando definido como 1, a SPI está ativada;
- DORD – Data Order – LSB transmitido primeiro, se 1, e MSB transmitido primeiro, se 0;
- MSTR – Master/Slave Select – Define o Arduino em Modo Mestre (Master Mode), se 1, e Modo Escravo (Slave Mode), quando 0;
- CPOL – Clock Polarity – Se 1, define o clock como ocioso quando em estado alto e, se 0, ocioso quando em estado baixo;
- CPHA – Clock Phase – Determina se a amostragem dos dados ocorre na extremidade ascendente ou descendente do clock;
- SPR1/0 – SPI Clock Rate Select 1 & 0 – Esses dois bits controlam a taxa do clock do dispositivo mestre.

É permitido que você altere essas configurações, pois dispositivos SPI diferentes esperam que a polaridade do clock, sua fase, a ordem dos dados, a velocidade etc. também sejam diferentes. Isso se deve principalmente ao fato de que não há uma configuração padrão para a SPI, portanto, fabricantes criam dispositivos distintos com pequenas diferenças. Em seu código, você definiu o SPCR da seguinte maneira:

```
SPCR = B01010011;
```

Dessa forma, você desativou a interrupção (SPIE = 0), ativou a SPI (SPE = 1), definiu a ordem dos dados como MSB primeiro (DORD = 0), o Arduino como master (MSTR = 1), a polaridade do clock como ociosa quando baixa (CPOL = 0), a fase do clock para que a amostragem seja feita na extremidade ascendente (CPHA = 0), e a velocidade como 250 kHz (SPR 1/0 = 11, 1/64 da frequência do oscilador do Arduino [16.000 / 64]).

O SPSR (SPI Status Register) utiliza três dos seus bits para definir o estado de transferência da SPI. Você está interessado apenas no bit 7 (SPIF – SPI Interrupt Flag, flag de interrupção da SPI), que avisa se uma transferência serial foi ou não concluída. Se uma transferência foi concluída, ele é definido como 1. Limpamos esse bit (definindo-o como 0) primeiro lendo o SPSR com o bit 7 definido como 1, e então acessamos o registro de dados da SPI (SPDR).

O SPDR simplesmente armazena o byte que será enviado na linha de comunicação MOSI e lido na linha MISO.

Todo esse processo pode parecer muito complicado, mas você não tem obrigatoriamente de sabê-lo por inteiro (ainda). O barramento da SPI pode ser explicado em termos práticos como tendo um dispositivo mestre e outro escravo, que desejam se comunicar. O pulso do clock assegura que os dados sejam enviados do mestre para o escravo quando o pulso sobe (ou desce, dependendo de como você configurou o registro de controle) e do escravo para o mestre quando o clock desce (ou sobe). A SPIF é definida como 1 depois que a transferência é concluída.

Agora, vamos retornar ao código.

Projeto 31 – Sensor digital de pressão – Análise do código (continuação)

Na sequência, você faz a leitura dos registros de SPSR e SPDR em `clr`. Isso garante a retirada de dados desnecessários existentes nesses registros, deixando o dispositivo pronto para o uso.

```
clr=SPSR;  // SPI Status Register (Registrador de estado da SPI)
clr=SPDR;  // SPI Data Register (Registrador de dados da SPI)
```

Depois, temos uma pequena espera, definimos a taxa de transferência e esperamos 500 milissegundos, para permitir que a linha serial se configure:

```
delay(10);
Serial.begin(38400);
delay(500);
```

Agora, você utiliza a função `write_register` (que será explicada mais adiante) para escrever dois valores no endereço de registrador de operação do sensor, o endereço hexadecimal `0x03`. Escrever um valor de `0x09` nesse registro definirá o sensor como modo de leitura de alta velocidade, e o envio do valor `0x0A` definirá o registrador como modo de aquisição de alta resolução, garantindo que você obtenha o valor mais preciso do sensor:

```
write_register(0x03,0x09);    // Modo de leitura de alta velocidade
write_register(0x03,0x0A);   // Modo de medição de alta resolução
```

Em seguida, temos o loop principal. Você inicia lendo o valor do registro PRESSURE (endereço 0x1F) e armazenando esse dado em pressure_msb. Esse valor será o byte mais significativo de seu valor de 19 bits (ou seja, os três bits de que você necessita):

```
pressure_msb = read_register(PRESSURE);
```

Em seguida, você transforma esse valor em uma máscara de bits, executando uma operação bit a bit E (&), que compara o valor em pressure_msb com o número B00000111. Lembre-se de que o operador bit a bit E define bits como 1 quando ambos os bits são 1, e como 0 quando ambos os bits são 0. Ter os três primeiros bits definidos como 1 (B00000111) significa que você terminará com apenas os três primeiros bits, exatamente o que pretendemos.

```
pressure_msb &= B00000111;
```

Agora você deseja obter os 16 bits remanescentes do valor de pressão, por isso lê o valor armazenado no registro PRESSURE_LSB (endereço 0x20):

```
pressure_lsb = read_register16(PRESSURE_LSB);
```

Então, você executa um E bit a bit, comparando esse valor com 0x0000FFFF, que corresponde ao número binário B1111111111111111. Isso significa que, ao final, você terá apenas os 16 primeiros bits de qualquer valor lido:

```
pressure_lsb &= 0x0000FFFF;
```

Depois, você utiliza o engenhoso truque da diretiva de definição UBLB19, para pegar os 16 bits e os três bits e combiná-los em um número de 19 bits:

```
pressure = UBLB19(pressure_msb, pressure_lsb);
```

O datasheet do SCP1000 diz que o valor da pressão é um inteiro; para convertê-lo de um inteiro decimal em um valor representado em pascals (medida de pressão), você tem de dividi-lo por quatro:

```
pressure /= 4;
```

Agora que você tem seus dados de pressão, deve enviá-los ao monitor serial, onde poderá lê-los. Portanto, você imprime o valor de pressão, primeiramente convertendo esse valor para hPa (hectopascals) dividindo o número por 100, e armazenando o resultado em um float, a variável hPa. É necessário primeiro convertê-lo em um float, para assegurar que não perderemos os dois dígitos depois da vírgula decimal.

```
Serial.print("Pressure (hPa): ");
float hPa = float(pressure)/100;
Serial.println(hPa);
```

Depois, você imprime a pressão novamente, mas, dessa vez, em atmosferas. Isso é feito dividindo o valor por 101.325. Uma atmosfera é igual a 101.325 pascals. O número 3 depois de pAtm, na terceira linha a seguir, diz ao monitor serial para imprimir o valor com três casas decimais.

```
Serial.print("Pressure (Atm): ");
float pAtm = float(pressure)/101325.0;
Serial.println(pAtm, 3);
```

Na sequência, você tem de ler os dados de temperatura. Isso é feito chamando a função `read_register16` e passando a ela o registro `TEMP` (endereço `0x21`). O valor retornado dessa função é armazenado na variável `temp_in`. A temperatura é um valor de 14 bits.

```
temp_in = read_register16(TEMP);
```

O valor é convertido em um `float` e dividido por `20`, para que possamos obter a temperatura em Celsius:

```
float tempC = float(temp_in)/20.0;
Serial.print("Temp. C: ");
Serial.println(tempC);
```

Depois, você simplesmente multiplica esse valor por 1,8 e adiciona 32 para obter a temperatura em Fahrenheit, emitindo também esses dados:

```
float tempF = (tempC*1.8) + 32;
Serial.print("Temp. F: ");
Serial.println(tempF);
```

Agora, você tem quatro funções que permitem a leitura de dados no barramento da SPI. A primeira é `spi_transfer`. Como você passará um `char` (um byte) a essa função e receberá um `char` de volta, ela é de tipo `char`. Os dados passados à função também são de tipo `char`.

```
char spi_transfer(char data)
```

O byte enviado para essa função é transmitido ao registro de dados da SPI:

```
SPDR = data;
```

Então, você espera até que a flag `SPIF` seja definida, para sinalizar que os dados foram enviados com êxito:

```
while (!(SPSR & (1<<SPIF))) { };
```

Não há nada dentro do bloco de código, por isso ele não fará nada até que SPIF seja definida. Isso é feito executando uma operação bit a bit com `SPSR` e o valor da flag `SPIF`:

```
!(SPSR & (1<<SPIF))
```

Vamos dividir essa operação em partes. Primeiramente, você tem:

```
(1<<SPIF)
```

Isso faz com que você mova o valor `1` na posição do sétimo bit, deixando-o pronto para ser uma máscara de bits. `SPIF` foi previamente definida como `7` em uma macro (para os chips Atmega).

Depois, você utiliza um E com essa máscara de bits e o valor em `SPSR`:

```
(SPSR & (1<<SPIF))
```

Isso resultará em um valor diferente de zero, se a flag `SPIF` estiver definida como `1`. Assim, se `SPSR` fosse, por exemplo, `B01010101`, e a flag `SPIF` fosse `1`, então o cálculo seria `(B01010101 & (1<<7))`, igual a `(B01010101 & B10000000)`, o que resultaria em `B00000000`, uma vez que o operador E resulta em valores 1 apenas quando os bits equivalentes de ambos os números forem 1.

Mas se `SPSR` for `B11010101` e a flag `SPIF` for `1`, então você tem `(B11010101) & (B10000000)`, o que resultará em `B10000000`, um valor diferente de zero. Todo o cálculo é conferido com o operador lógico NÃO (`!`):

```
!(SPSR & (1<<SPIF))
```

Em outras palavras, se a flag `SPIF` NÃO (`!`) estiver definida como `1`, não faça nada, e continue a não fazer nada até que a flag `SPIF` esteja definida como `1`, o que fará com que o fluxo do código saia do loop `while` e execute o comando a seguir:

```
return SPDR;
```

Isso retorna o valor no registrador `SPDR`. Assim, a função como um todo envia um byte do mestre para o escravo, espera até que isso seja feito, e retorna um byte do escravo para o mestre.

A função seguinte é usada para ler um valor de 8 bits a partir de um registrador no SCP1000. Ela pega um `char` como parâmetro e retorna um `char`:

```
char read_register(char register_name)
```

Uma variável de tipo `char` é declarada e nomeada `in_byte`. Ela armazenará o valor lido no registro de pressão:

```
char in_byte;
```

Depois, o endereço do registro é deslocado duas posições para a esquerda:

```
register_name <<= 2;
```

Agora você ativa o dispositivo SPI, colocando a linha do Slave Select (CSB) no estado baixo:

```
digitalWrite(SLAVESELECT, LOW); // Ativa o dispositivo SPI
```

Em seguida, você transfere o nome do registro para o dispositivo:

```
spi_transfer(register_name);    // Escreve o byte no dispositivo
```

Depois, você envia outro byte, mas, dessa vez, não envia nada, por isso recebe o valor do registro de volta:

```
in_byte = spi_transfer(0x00);   // Não envia nada, mas pega de volta o valor do registro
```

`in_byte` agora armazena o valor MSB da leitura de pressão. Na sequência, a linha SPI é desativada, e você retorna o MSB da leitura de pressão para o loop principal:

```
digitalWrite(SLAVESELECT, HIGH);  // Desativa o dispositivo SPI
delay(10);
return(in_byte); // valor de retorno
```

A próxima função é `read_register16()`, que executa praticamente a mesma ação, exceto que, dessa vez, em vez de retornar um byte de 8 bits, retorna uma palavra de 16 bits. Novamente, o endereço do registro é transmitido à função.

```
unsigned long read_register16(char register_name)
```

Então, você declara dois `bytes` e um `float`:

```
byte in_byte1;
byte in_byte2;
float in_word;
```

Os dois bytes armazenarão, cada um, oito bits da palavra de 16 bits, e o `float` armazenará a palavra final de 16 bits que a função transmitirá de volta. O restante da função é idêntico à implementação anterior, exceto que, dessa vez, você recebe de volta dois conjuntos de 8 bits, em vez de apenas um.

```
register_name <<= 2;
digitalWrite(SLAVESELECT, LOW);  // Ativa o dispositivo
spi_transfer(register_name);     // Escreve o byte para o dispositivo
in_byte1 = spi_transfer(0x00);
in_byte2 = spi_transfer(0x00);
digitalWrite(SLAVESELECT, HIGH);   // Desativa o dispositivo
```

O comando `UBLB` pega os dois bytes e, utilizando deslocamentos, os transforma em uma palavra de 16 bits. O resultado é, então, retornado pela função:

```
in_word = UBLB(in_byte1,in_byte2);
return(in_word); // valor de retorno
```

O propósito da última função é escrever um valor em um registrador no SCP1000. A função não retorna nada, por isso é de tipo `void` e tem dois parâmetros: o endereço do registrador e o valor a ser escrito:

```
void write_register(char register_name, char register_value)
```

O endereço do registrador é deslocado duas posições para a esquerda. Depois, aplicamos um OU bit a bit entre ele e `B00000010`, para criar o comando de escrita:

```
register_name <<= 2;
register_name |= B00000010;  // Comando de escrita
```

A linha SPI é ativada, o endereço do registrador e seu valor são transferidos para o dispositivo e a linha SPI é fechada novamente, antes que a função seja encerrada:

```
digitalWrite(SLAVESELECT, LOW);      // Seleciona o dispositivo SPI
spi_transfer(register_name);        // Envia a localização do registrador
spi_transfer(register_value);       // Envia o valor a ser gravado no registrador
digitalWrite(SLAVESELECT, HIGH);
```

O.k., admitimos que utilizar a SPI não é algo muito simples, mas esperamos que, depois do que mostramos, você ao menos compreenda como ela funciona. Este projeto está simplesmente acessando leituras de temperatura e pressão, e enviando-as para o monitor serial. Como o SCP1000 é um dispositivo fechado, não farei uma análise do hardware. Tudo que você tem de saber é que ele lê dados de pressão e temperatura e transmite essas informações pela linha serial da SPI.

Agora, vamos encontrar uma aplicação prática para leituras de pressão.

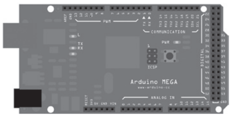
Projeto 32 – Barógrafo digital

Agora que você pode conectar o SCP1000 e obter seus dados, vamos utilizá-lo na prática. Este projeto criará um barógrafo digital, um gráfico de pressão sobre tempo. Para exibir o gráfico, você utilizará um GLCD (Graphic LCD). Neste projeto você aprenderá como exibir gráficos, assim como texto.

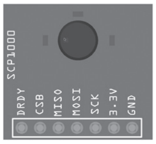
Componentes necessários

A lista de componentes deste projeto é idêntica à do projeto 31, com a adição de um GLCD de 128 x 64, um resistor extra e um potenciômetro. Você utilizará a biblioteca `glcd.h`, por isso o GLCD deve ter um chip controlador KS0108 (ou equivalente). Verifique seu datasheet antes da compra.

Arduino Mega



Sensor de pressão SCP1000



3 resistores de 10 kΩ



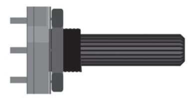
1 resistor de 1 kΩ



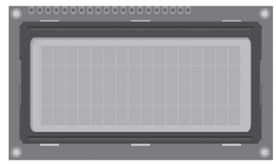
1 resistor de 150 Ω



Potenciômetro de 10 kΩ



GLCD de 128 x 64



Conectando os componentes

Conecte tudo como na figura 11.5.

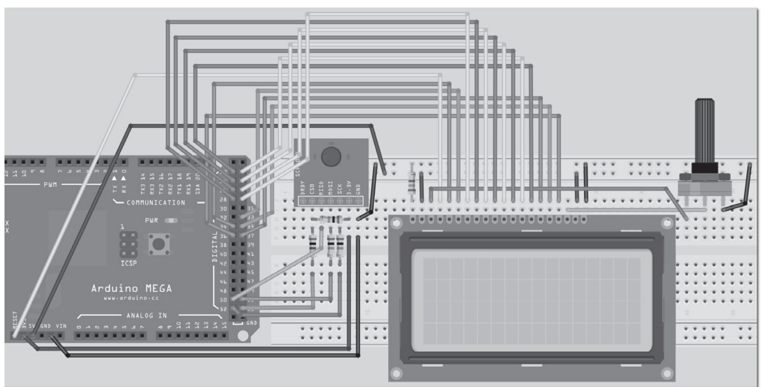


Figura 11.5 – Circuito para o Projeto 32 – Barógrafo digital (consulte o site da Novatec para versão colorida).

A seção do circuito correspondente ao SCP1000 não foi alterada com relação ao projeto 31. Você está simplesmente adicionando alguns outros componentes ao circuito. O GLCD que você vai utilizar pode ter uma pinagem diferente da mostrada neste projeto. Leia seu datasheet e certifique-se de que os pinos correspondam àqueles da tabela 11.2.

Tabela 11.2 – Pinagem entre o Mega e o GLCD

Mega	GLCD	Outro
Terra (GND)	Terra (GND)	
+5 V	+5 V	
	Contraste do LCD	Pino central do potenciômetro
36	D/I	
35	R/W	
37	Enable	
22	DB0	

Mega	GLCD	Outro
23	DB1	
24	DB2	
25	DB3	
26	DB4	
27	DB5	
28	DB6	
29	DB7	
33	CS1	
34	CS2	
Reset	Reset	
	VEE	Lado positivo do potenciômetro
	+5 V da backlight	+5 V
	+0 V da backlight	Terra (GND) via resistor de 150 Ω

Não ligue o circuito até que tudo esteja devidamente conectado e que você tenha verificado calmamente sua fiação. É muito fácil danificar um GLCD ao conectar sua fiação da forma errada. Em específico, certifique-se de que o potenciômetro tenha um de seus lados conectado ao terra, o pino do centro ao pino de ajuste de contraste do LCD, e o outro pino ao VEE (Voltagem do LCD). O resistor de 150 Ω serve para limitar a corrente que vai para a backlight do LCD; talvez você tenha de alterar esse valor para obter o brilho correto, mas certifique-se de não exceder a voltagem mostrada no datasheet para seu LCD. O potenciômetro é utilizado para ajustar o contraste do display, de modo a facilitar sua visualização.

Assim que você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino. Você deve ver o GLCD acender, pronto para receber dados. Agora, digite o código.

Digite o código

Antes de digitar o código, será necessário fazer o download e instalar a biblioteca `glcd.h`. Essa incrível biblioteca, escrita por Michael Margolis, está atualmente em sua terceira versão e vem acompanhada de um ótimo documento, que mostra como conectar tipos diferentes de GLCDs, além de instruções completas para todos os seus comandos. Consulte o Arduino Playground, na área de LCDs, para encontrar esse texto. Assim que você tiver feito o download da biblioteca, descompacte-a e coloque a pasta inteira resultante no diretório "libraries", dentro de sua instalação do Arduino. Da próxima vez que você ligar o IDE de seu Arduino, ela estará carregada e pronta para ser utilizada.

Digite o código da listagem 11.2.

Listagem 11.2 – Código para o projeto 32

```
/*
SCP1000    Mega
DRDY       N/D
CSB        53 via conversor de nível lógico
MISO       50 (direto)
MOSI       51 via conversor de nível lógico
SCK        52 via conversor de nível lógico
3.3 V      3.3 V
GND        Terra (GND)
TRIG       Terra (GND)
PD         Terra (GND)
*/

#include <glcd.h>
#include "fonts/allFonts.h"

// Pinos da SPI
#define SLAVESELECT 53
#define SPICLOCK 52
#define DATAOUT 51    // MOSI
#define DATAIN 50     // MISO
#define UBLB(a,b)  ( ( (a) << 8) | (b) )
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )

// Endereços
#define PRESSURE 0x1F        // 3 bits mais significativos da pressão
#define PRESSURE_LSB 0x20    // 16 bits menos significativos da pressão
#define TEMP 0x21            // 16 bits da temperatura
#define INTERVAL 900         // Intervalo de tempo em segundos (aproximado)

int dots[124], dotCursor = 0, counter = 0;
char rev_in_byte;
int temp_in;
float hPa;
unsigned long pressure_lsb;
unsigned long pressure_msb;
unsigned long temp_pressure;
unsigned long pressure;

void setup() {
  GLCD.Init();         // inicializa a biblioteca
  GLCD.ClearScreen();
  GLCD.SelectFont(System5x7, BLACK);    // carrega a fonte
  byte clr;
```

```
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH);   // desativa o dispositivo

  SPCR = B01010011;   // SPI Control Register (Registrador de controle da SPI)
  // MPIE=0, SPE=1 (ativado), DORD=0 (MSB primeiro), MSTR=1 (mestre), CPOL=0 (clock ocioso
  // quando em estado baixo), CPHA=0 (faz a amostragem do MOSI na extremidade ascendente),
  // SPR1=1 & SPR0=1 (250 kHz)
  clr=SPSR;       // SPI Status Register (Registrador de estado da SPI)
  clr=SPDR;       // SPI Data Register (Registrador de dados da SPI)
  delay(10);

  write_register(0x03,0x09);   // Modo de leitura de alta velocidade
  write_register(0x03,0x0A);   // Modo de medição de alta resolução

  GLCD.DrawRect(1,1,125,44);   // Desenha um retângulo
  for (int x=0; x<46; x+=11) {   // Desenha as linhas de escala vertical
    GLCD.SetDot(0,1+x, BLACK);
    GLCD.SetDot(127,1+x, BLACK);
  }
    for (int x=0; x<128; x+=5) {   // Desenha as linhas de escala horizontal
    GLCD.SetDot(1+x,0, BLACK);
  }

  for (int x; x<124; x++) {dots[x]=1023;}   // limpa o array
  getPressure();
  drawPoints(dotCursor);
}

void loop() {
  getPressure();

  GLCD.CursorToXY(0, 49);   // imprime a pressão
  GLCD.print("hPa:");
  GLCD.CursorToXY(24,49);
  GLCD.print(hPa);

  temp_in = read_register16(TEMP);
  float tempC = float(temp_in)/20.0;
  float tempF = (tempC*1.8) + 32;

  GLCD.CursorToXY(0,57);   // imprime a temperatura
  GLCD.print("Temp:");
  GLCD.CursorToXY(28, 57);
  GLCD.print(tempC);     // muda para tempF, caso queira Fahrenheit

  delay(1000);
```

```
  GLCD.CursorToXY(84,49);    // imprime a tendência
  GLCD.print("TREND:");
  GLCD.CursorToXY(84,57);
  printTrend();

   counter++;
  if (counter==INTERVAL) {drawPoints(dotCursor);}

}

void drawPoints(int position) {
  counter=0;
  dots[dotCursor] = int(hPa);
  GLCD.FillRect(2, 2, 123, 40, WHITE);       // limpa a área do gráfico
  for (int x=0; x<124; x++) {
    GLCD.SetDot(125-x,44-((dots[position]-980)), BLACK);
    position--;
    if (position<0) {position=123;}
  }
  dotCursor++;
  if (dotCursor>123) {dotCursor=0;}
  }

void getPressure() {
  pressure_msb = read_register(PRESSURE);
  pressure_msb &= B00000111;
  pressure_lsb = read_register16(PRESSURE_LSB);
  pressure_lsb &= 0x0000FFFF;
  pressure = UBLB19(pressure_msb, pressure_lsb);
  pressure /= 4;
  hPa = float(pressure)/100;
}

void printTrend() {      // calcula a tendência desde o último ponto de dados e imprime
  int dotCursor2=dotCursor-1;
  if (dotCursor2<0) {dotCursor2=123;}
  int val1=dots[dotCursor2];
  int dotCursor3=dotCursor2-1;
  if (dotCursor3<0) {dotCursor3=123;}
  int val2=dots[dotCursor3];
  if (val1>val2) {GLCD.print("RISING ");}
  if (val1==val2) {GLCD.print("STEADY ");}
  if (val1<val2) {GLCD.print("FALLING");}
}
```

```
char spi_transfer(char data) {
  SPDR = data;             // Inicia a transmissão
  while (!(SPSR & (1<<SPIF)))     // Aguarda o fim da transmissão
  {
  };
  return SPDR;    // retorna o byte recebido
}

char read_register(char register_name) {
    char in_byte;
    register_name <<= 2;

    digitalWrite(SLAVESELECT,LOW);   // Ativa o dispositivo SPI
    spi_transfer(register_name);     // Escreve o byte no dispositivo
    in_byte = spi_transfer(0x00);    // Não envia nada, mas pega de volta o valor do registro
    digitalWrite(SLAVESELECT,HIGH);  // Desativa o dispositivo SPI
    delay(10);
    return(in_byte);        // valor de retorno
}

unsigned long read_register16(char register_name) {
    byte in_byte1;
    byte in_byte2;
    float in_word;

    register_name <<= 2;

    digitalWrite(SLAVESELECT,LOW);   // Ativa o dispositivo SPI
    spi_transfer(register_name);     // Escreve o byte para o dispositivo
    in_byte1 = spi_transfer(0x00);
    in_byte2 = spi_transfer(0x00);
    digitalWrite(SLAVESELECT,HIGH);  // Desativa o dispositivo SPI
    in_word = UBLB(in_byte1,in_byte2);
    return(in_word);        // valor de retorno
}

void write_register(char register_name, char register_value) {
    register_name <<= 2;
    register_name |= B00000010;    // Comando de escrita

    digitalWrite(SLAVESELECT,LOW);   // Seleciona o dispositivo SPI
    spi_transfer(register_name);     // Envia a localização do registrador
    spi_transfer(register_value);    // Envia o valor para gravar no registrador
    digitalWrite(SLAVESELECT,HIGH);
}
```

Ao executar o código, você verá um gráfico de pressão ao longo do tempo, assim como uma medição atual de pressão e temperatura. À primeira vista, isso não é muito empolgante, uma vez que são necessárias 24 horas para mostrar as variações na pressão. Cada ponto representa 15 minutos de tempo, e a escala horizontal está definida em horas. Caso você queira preencher o gráfico mais rapidamente, altere o valor no define de `INTERVAL` para um número menor.

Projeto 32 – Barógrafo digital – Análise do código

A maioria do código para o projeto 32 é idêntica ao código do projeto 31. Entretanto, incluímos seções novas para controlar o GLCD, e removemos os trechos para envio de dados ao monitor serial. Portanto, abordaremos rapidamente o código repetido, concentrando nossa atenção nas novas inclusões.

Primeiro, você tem de incluir a biblioteca `glcd.h` em seu código, assim como as fontes que você utilizará:

```
#include <glcd.h>
#include "fonts/allFonts.h"
```

Na seção de endereços, há um novo `#define` para `INTERVAL`, que define o intervalo, em segundos, entre os pontos de dados armazenados e exibidos no gráfico. O intervalo é de 900 segundos, o que representa 15 minutos entre os pontos.

```
#define INTERVAL 900    // Intervalo de tempo em segundos (aproximado)
```

Três novos inteiros são declarados e inicializados. Um é o array `dots[]`, que armazena as medições de pressão lidas em intervalos de 15 minutos. O próximo é a variável `dotCursor`, que armazena o índice do array que você está atualmente utilizando. Por fim, a variável `counter` será incrementada sempre que o loop principal se repetir, sendo utilizada para ver quantos segundos (aproximadamente) transcorreram desde que o último ponto de dados foi armazenado.

```
int dots[124], dotCursor = 0, counter = 0;;
```

Na rotina de inicialização, você primeiro inicializa o dispositivo GLCD:

```
GLCD.Init();
```

Os comandos para controlar o GLCD vêm todos depois do ponto, como no comando que vemos a seguir, utilizado para limpar a tela:

```
GLCD.ClearScreen();
```

Na sequência, você escolhe qual fonte utilizará, e se ela deve ser exibida utilizando pixels pretos ou brancos:

```
GLCD.SelectFont(System5x7, BLACK);   // carrega a fonte
```

Há muitos tipos e tamanhos diferentes de fontes que podem ser utilizados. Da mesma forma, a biblioteca inclui um interessante software gratuito, o FontCreator2, que pode ser utilizado para criar um arquivo de cabeçalho de fonte a ser incluído em seu sketch. Esse programa pode converter fontes do PC, para que possam ser utilizadas com a biblioteca.

Na sequência, exibimos a caixa do gráfico com o comando `DrawRect`:

```
GLCD.DrawRect(1,1,125,44);
```

O sistema de coordenadas para o display de 128 x 64 é de 128 pixels de largura e 64 pixels de altura, tendo o pixel 0 para ambos os eixos no canto superior esquerdo. A coordenada X vai de 0 a 127, e a coordenada Y vai de 0 (topo) a 63 (base). `DrawRect` desenha um retângulo a partir das coordenadas X e Y, que formam os dois primeiros parâmetros, e que representam o canto superior esquerdo da caixa. Os dois parâmetros seguintes são a altura e largura, que se estendem a partir das coordenadas X e Y indicadas nos dois primeiros parâmetros. Sua caixa começa no ponto (1, 1) e se estende por 125 pixels de largura e 44 pixels de altura.

Você também pode criar um retângulo preenchido com o comando `FillRect()`, da seguinte maneira:

```
GLCD.FillRect(1,1,125,44, BLACK);
```

Isso criará um retângulo sólido preto com as mesmas dimensões do anterior. Em seguida, você necessita das linhas de escala, verticais e horizontais. Você utiliza um loop `for` para criar pontos na vertical a intervalos de 11 pixels, usando as coordenadas 0 e 127 no eixo X, partindo do pixel 1 no eixo Y:

```
 for (int x=0; x<46; x+=11) {
  GLCD.SetDot(0,1+x, BLACK);
  GLCD.SetDot(127,1+x, BLACK);
}
```

Isso é feito com o comando `SetDot`, que simplesmente coloca um único pixel nas coordenadas X e Y, em `BLACK` (preto) ou `WHITE` (branco). Pixels brancos simplesmente apagarão pixels pretos já exibidos.

Depois, desenhamos a escala horizontal (horas), com intervalos de cinco pixels, a partir do pixel 1 no eixo X, em direção ao lado direito:

```
for (int x=0; x<128; x+=5) {
  GLCD.SetDot(1+x,0, BLACK);
}
```

Na sequência, o array das medições de pressão é inicializado com o valor de `1023`. Isso é feito porque o gráfico exibirá todos os pontos armazenados desde o início. Como você não deseja exibir todos os pontos, mas apenas aqueles armazenados desde que

o Arduino foi ligado, o valor de `1023` assegurará que essas medições coincidam com a linha do topo do gráfico e que fiquem, portanto, escondidas. Explicaremos isso melhor na função `drawPoints()`.

```
for (int x; x<124; x++) {dots[x]=1023;}
```

Depois, você chama a função `getPressure()`. O código no loop principal do projeto 31 foi simplesmente copiado para a função `getPressure()`, uma vez que ela é chamada algumas vezes no código. As seções do projeto 31 que enviavam dados para o monitor serial foram todas removidas, uma vez que agora os dados serão exibidos no GLCD.

```
getPressure();
```

Agora você chama a função `drawPoints()`:

```
drawPoints(dotCursor);
```

Essa é uma função para desenhar pontos no gráfico, e será explicada em breve. Você obtém a pressão e desenha o gráfico antes de executar o loop principal, para que já exista uma medição armazenada no primeiro índice do array `dots[]`. Isso é vital para garantir que a função `printTrend()` (a ser explicada em breve) funcione corretamente.

No loop principal do programa, você encontra a pressão atual:

```
getPressure();
```

Então, você imprime essa pressão no display. Para imprimir texto, o cursor deve primeiro ser movimentado para a posição correta. Para isso, colocamos uma coordenada X e Y no comando `CursorToXY(x,y)` e, depois, imprimimos o texto apropriado. Fazemos isso duas vezes para imprimir a palavra "hPa:", seguida pelo valor de pressão.

```
GLCD.CursorToXY(0, 49);
GLCD.print("hPa:");
GLCD.CursorToXY(24,49);
GLCD.print(hPa);
```

O mesmo é feito para imprimir a temperatura, logo abaixo da pressão:

```
GLCD.CursorToXY(0,57);
GLCD.print("Temp:");
GLCD.CursorToXY(28, 57);
GLCD.print(tempC);
```

Caso você queira exibir a temperatura em Fahrenheit, em vez de Celsius, altere `tempC` para `tempF`.

Depois, temos uma espera de mil milissegundos, o que significa que a pressão será obtida e exibida aproximadamente a cada segundo:

```
delay(1000);
```

Na sequência, imprimimos a tendência da pressão. O valor da tendência é obtido chamando a função `printTrend()` (que será explicada em breve):

```
GLCD.CursorToXY(84,49);
GLCD.print("TREND:");
GLCD.CursorToXY(84,57);
printTrend();
```

Você deseja apenas armazenar o valor atual da pressão a cada intervalo específico (`INTERVAL`) de segundos. Assim, depois de cada espera de mil milissegundos, você incrementa `counter` em 1:

```
counter++;
```

E verifica se o valor de `counter` atingiu o valor de `INTERVAL`. Se afirmativo, você chama a função `drawPoints()`:

```
if (counter==INTERVAL) {drawPoints(dotCursor);}
```

Na sequência, adicionamos duas novas funções para desenhar o gráfico e imprimir a tendência atual de pressão. A primeira é a função `drawPoints()`. Você passa a ela o valor de `dotCursor` como parâmetro:

```
void drawPoints(int position) {
```

O valor de `counter` atingiu o de `INTERVAL`, por isso você o redefine como 0:

```
counter=0;
```

A leitura de pressão atual é armazenada no array `dots[]`, na posição atual. Como você está interessado apenas em um ponto a ser exibido em um display de baixa resolução, não são necessários números depois da vírgula decimal, por isso converta `hPa` em um inteiro. Isso também ajuda a economizar memória, uma vez que um array de inteiros ocupa menos espaço que um array de floats.

```
dots[dotCursor] = int(hPa);
```

Agora você tem de limpar o gráfico, deixando-o pronto para os novos dados. Isso é feito com um comando `FillRect`, que cria um retângulo limitado pelas bordas da caixa do gráfico:

```
GLCD.FillRect(2, 2, 123, 40, WHITE);   // limpa a área do gráfico
```

Em seguida, você itera os 124 elementos do array com um loop `for`:

```
for (int x=0; x<124; x++) {
```

E posiciona um ponto na posição apropriada do gráfico, utilizando o comando `SetDot()`:

```
GLCD.SetDot(125-x,44-((dots[position]-980)), BLACK);
```

Você deseja que o gráfico seja desenhado da direita para a esquerda, de modo que a leitura atual esteja no lado direito e o gráfico se estenda para a esquerda. Assim,

você inicia desenhando o primeiro ponto na coordenada X de valor `125-x`, o que fará com que os pontos se movimentem para a esquerda conforme se eleva o valor de x. A coordenada Y é determinada pegando a pressão em hPa e subtraindo `980`. A escala vertical do gráfico tem 40 pixels de altura, alcançando os valores hPa de 980 a 1.020.

Esses são valores típicos de hPa para a maioria das localidades. Se você vive em uma região com valores de pressão geralmente mais altos ou baixos, pode ajustar o valor de 980 de acordo com as suas necessidades. Depois, subtraia esse número de 44, obtendo a posição Y do ponto para essa leitura específica de pressão no array.

O valor de `position`, originalmente definido como o valor atual de `dotCursor`, é decrementado:

```
position--;
```

Caso esse valor desça abaixo de zero, ele será novamente definido como `123`:

```
if (position<0) {position=123;}
```

Assim que todos os 124 pontos tiverem sido desenhados, o valor de `dotCursor` será incrementado, ficando pronto para armazenar a próxima medição de pressão no array:

```
dotCursor++;
```

Caso ultrapasse o valor de 123 (elemento máximo do array), `dotCursor` será novamente definido como zero:

```
if (dotCursor>123) {dotCursor=0;}
```

A próxima função é `printTrend()`, cujo trabalho é simplesmente descobrir se a medição atual de pressão armazenada é maior, menor ou igual à última leitura armazenada e imprimir `RISING` (crescente), `STEADY` (constante) ou `FALLING` (decrescente), de acordo.

Você inicia armazenando a última posição de `dotCursor` em `dotCursor2`. Você subtrai 1 de seu valor, uma vez que ele foi incrementado depois que a medição foi armazenada na função `drawPoints()`.

```
int dotCursor2=dotCursor-1;
```

Você verifica se esse valor é menor que zero, definindo-o, se afirmativo, novamente como `123`:

```
if (dotCursor2<0) {dotCursor2=123;}
```

`dotCursor2` agora armazena a posição no array da última medição feita. Você declara um inteiro `val1` e armazena nele a última medição de pressão:

```
int val1=dots[dotCursor2];
```

Agora você quer a medição tomada ANTES da última medição, por isso cria outra variável, `dotCursor3`, que armazenará a posição no array antes da última medição tomada:

```
int dotCursor3=dotCursor2-1;
if (dotCursor3<0) {dotCursor3=123;}
int val2=dots[dotCursor3];
```

Dessa forma, você tem `val1` com a última leitura da pressão, e `val2` com a leitura antes dela. Tudo que falta fazer é decidir se a última leitura de pressão tomada é maior, menor ou a mesma da anterior, e imprimir a tendência relevante de acordo.

```
if (val1>val2) {GLCD.print("RISING ");}
if (val1==val2) {GLCD.print("STEADY ");}
if (val1<val2) {GLCD.print("FALLING");}
```

O restante do código é idêntico ao projeto 31. Quando você executar este programa, verá um display semelhante ao da figura 11.6. Se a pressão tiver passado por alterações consideráveis nas últimas 24 horas, você verá uma linha com grandes variações.

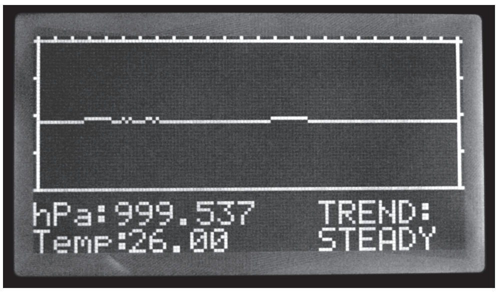


Figura 11.6 – Display para o projeto 32 – Barógrafo digital.

O propósito principal do projeto 32 foi mostrar-lhe uma utilização prática do sensor SCP1000 e como utilizar um display GLCD. Os comandos apresentados para o GLCD foram apenas uma amostra do que pode ser feito com ele. Você pode exibir retângulos, círculos e linhas, definir pontos e até mesmo desenhar bitmaps com um GLCD. A tela pode ser dividida em áreas de texto e de gráficos, que podem ser acessadas independentemente. A biblioteca vem acompanhada de uma documentação muito boa, criada por Michael Margolis, sendo muito fácil utilizá-la. Leia com calma o texto, e você descobrirá que muito pode ser feito com ela. A biblioteca também vem acompanhada de um conjunto completo de sketches de exemplo, incluindo o Game of Life, de Conway, e um divertido joguinho de foguetes.

Resumo

O capítulo 11 mostrou-lhe como utilizar um sensor digital de pressão e como se comunicar com ele utilizando uma interface periférica serial (SPI). Você foi apresentado aos conceitos básicos da SPI e como ela funciona. Agora você tem uma boa noção de seu funcionamento, e sabe que pode utilizar a ótima biblioteca SPI que acompanha a versão 0019 da biblioteca do Arduino. Sabe também que essa biblioteca cuidará de

todo o trabalho pesado quando estivermos nos comunicando com outros dispositivos SPI. Você também aprendeu como utilizar a SPI para ler dados do pequeno e incrível sensor de pressão, o SCP1000. Caso queira criar sua própria estação meteorológica, esse econômico sensor é a escolha ideal. Em meu caso, escolhi esse sensor na criação de um projeto de balão de alta altitude por causa de seu pequeno tamanho, de sua precisão e facilidade de uso. Além de tudo isso, ele também fornece leituras de pressão abaixo do intervalo especificado no datasheet, o que o torna ideal para experimentos com HABs (High Altitude Balloons, balões de alta altitude).

Você também aprendeu como conectar um LCD gráfico ao Arduino, e como é fácil imprimir textos e gráficos básicos nele, utilizando a biblioteca `glcd.h`. Lendo mais da documentação, você pode aprender como fazer outras coisas interessantes, como exibir bitmaps ou criar seus próprios jogos. Um Arduino com um GLCD pode facilmente ser colocado dentro de uma pequena caixa, para criar seu próprio console de jogos portátil.

No próximo capítulo, você aprenderá como utilizar uma tela de toque.

Assuntos e conceitos abordados no capítulo 11:

- como conectar um sensor de pressão SCP1000 a um Arduino;
- como utilizar um `#define` para executar operações bit a bit em um conjunto de números;
- como criar números de maior comprimento de bits, combinando números de comprimentos menores;
- como funciona uma interface SPI;
- dispositivos SPI podem ser controlados separadamente ou encadeados;
- um dispositivo SPI é formado por um mestre e um escravo;
- como os dados entram e saem de um dispositivo SPI de acordo com o pulso do clock;
- o propósito e uso dos três registradores de barramento da SPI;
- como converter valores de pressão em pascals para hectopascals e atmosferas;
- como utilizar operadores bit a bit para verificar se um único bit está definido ou não;
- como conectar um LCD gráfico ao Arduino;
- como utilizar os comandos da biblioteca `glcd.h` para desenhar linhas, pontos e imprimir texto.

CAPÍTULO 12

Tela de toque

Neste capítulo, você conhecerá um dispositivo muito interessante, que pode ser utilizado facilmente com um Arduino — uma tela de toque, ou tela sensível ao toque (*touchscreen*). Com o crescimento da popularidade dos smartphones e consoles portáteis, telas de toque tornaram-se baratas e amplamente disponíveis. Uma tela desse tipo permite que você crie uma interface de toque para um dispositivo, ou pode ser colocada sobre uma tela LCD, para oferecer uma interface de toque. Empresas como a Sparkfun facilitam a capacidade de interfacear com esses dispositivos, fornecendo conectores e placas breakout. Uma *placa breakout*[1] é uma placa de circuito impresso que permite a você pegar o que normalmente seria um pequeno conjunto de conectores, ou um conector não padronizado, e utilizar esses componentes de modo mais prático, empregando elementos como pinos de cabeçalho, que podem ser utilizados para conectar a placa a uma protoboard. Neste projeto, você utilizará uma tela de toque do Nintendo DS com uma placa breakout da Sparkfun. Você iniciará com um simples projeto que mostra como obter leituras da tela de toque antes de utilizá-la.

Projeto 33 – Tela de toque básica

Para este projeto, você deve adquirir uma tela de toque do Nintendo DS, assim como uma placa breakout. Esta última é essencial, uma vez que a saída da tela de toque é um conector de fita muito fino e frágil, e será impossível fazer a interface com o Arduino sem componentes adicionais.

Componentes necessários

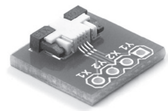
Telas de toque do Nintendo DS podem ser adquiridas a preços baixos em muitos fornecedores. A versão XL tem praticamente o dobro do tamanho da versão padrão; essa é a unidade recomendada, se você puder adquiri-la. A placa breakout foi adquirida na Sparkfun, mas também pode ser adquirida em outros distribuidores.

1 N.T.: Break out significa "escapar" ou "fugir"; assim, dizer que uma placa é "breakout" significa que ela foi "destacada" ou "separada", para servir de interface e facilitar o trabalho com determinados componentes.

Tela de toque do Nintendo DS



Breakout para tela de toque



Conectando os componentes

Conecte tudo como na figura 12.1.

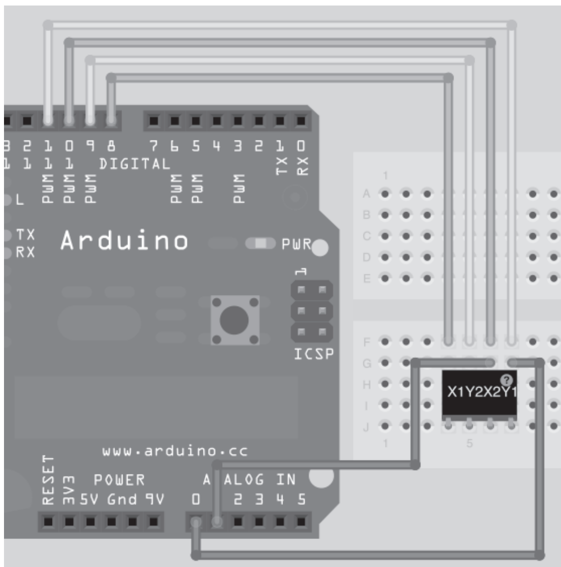


Figura 12.1 – Circuito para o Projeto 33 – Tela de toque básica (consulte o site da Novatec para versão colorida).

A placa breakout tem pinos marcados como X1, Y2, X2 e Y1. Conecte os pinos da forma descrita na tabela 12.1.

Tabela 12.1 – Conexões dos pinos para o projeto 33

Arduino	Breakout
Pino digital 8	X1
Pino digital 9	Y2
Pino digital10	X2
Pino digital 11	Y1
Pino analógico 0	Y1
Pino analógico 1	X2

Você terá de soldar alguns pinos de cabeçalho à unidade de breakout. Os pinos são soldados com o logotipo da Sparkfun virado para cima. A tela é conectada à placa breakout por meio de um pequeno conector. Puxe a aba e insira o pequeno cabo de fita no conector, então feche a aba para prendê-lo. A tela dever ser utilizada de modo que o conector em fita fique no canto superior direito. Daqui em diante, tenha cuidado com a placa, uma vez que ela é muito frágil e pode se quebrar com facilidade. Quebrei três telas e duas placas breakouts durante os testes. Se você encontrar um modo de fixar a protoboard, a placa breakout e a tela de toque, para impedir que se movimentem, faça isso.

Digite o código

Digite o código da listagem 12.1.

Listagem 12.1 – Código para o projeto 33

```
// Projeto 33

// Conexões de alimentação
#define Left 8   // Esquerda (X1) para o pino digital 8
#define Bottom 9 // Base (Y2) para o pino digital 9
#define Right 10 // Direita (X2) para o pino digital 10
#define Top 11   // Topo (Y1) para o pino digital 11

// Conexões analógicas
#define topInput 0    // Topo (Y1) para o pino analógico 0
#define rightInput 1  // Direita (X2) para o pino analógico 1

int coordX = 0, coordY = 0;

void setup()
{
  Serial.begin(38400);
}

void loop()
{
  if (touch())    // Se a tela foi tocada, imprime as coordenadas
  {
    Serial.print(coordX);
    Serial.print("  ");
    Serial.println(coordY);
    delay(250);
  }
}
```

```
// retorna true se tocada, e define as coordenadas touchX e touchY
boolean touch()
{
  boolean touch = false;

  // pega as coordenadas horizontais
  pinMode(Left, OUTPUT);
  digitalWrite(Left, LOW);  // Define o lado esquerdo para o terra

  pinMode(Right, OUTPUT);   // Defina o lado direito para os +5 V
  digitalWrite(Right, HIGH);

  pinMode(Top, INPUT);     // Topo e base com impedância alta
  pinMode(Bottom, INPUT);

  delay(3);
  coordX = analogRead(topInput);

  // pega as coordenadas verticais
  pinMode(Bottom, OUTPUT);  // Define a base para o terra
  digitalWrite(Bottom, LOW);

  pinMode(Top, OUTPUT);    // Defina o topo para os +5 V
  digitalWrite(Top, HIGH);

  pinMode(Right, INPUT);   // Esquerda e direita com impedância alta
  pinMode(Left, INPUT);

  delay(3);
  coordY = analogRead(rightInput);

  // se as coordenadas lidas forem menores do que 1000 e maiores do que 0, a tela foi tocada

  if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}

    return touch;
}
```

Digite o código e faça seu upload para o Arduino. Assim que ele estiver sendo executado, abra o monitor serial e então toque na tela de toque. Sempre que a tela for tocada, as coordenadas de seu dedo serão exibidas no monitor serial. As coordenadas são X no plano horizontal, da esquerda para a direita, e Y no plano vertical, do topo para a base.

Antes que você possa analisar o código, será interessante compreender como funciona uma tela de toque. Faremos uma breve análise do hardware antes de verificar o código.

Projeto 33 – Tela de toque básica – Análise do hardware

A tela de toque que você está utilizando neste projeto, típica de um Nintendo DS, é conhecida como *tela de toque resistiva*. Trata-se de uma construção relativamente simples, formada de diferentes camadas. A camada da base é de vidro, revestida por uma película transparente de óxido de metal. Isso torna o revestimento tanto condutivo quanto resistivo. Uma tensão aplicada à película apresenta um gradiente. Sobre a camada rígida de vidro temos uma camada superior flexível, também coberta pela película transparente resistiva. Essas duas camadas são separadas por um espaço muito pequeno, formado por uma grade de pequeninos pontos isolantes, com a função de manter separadas as duas camadas condutivas, impedindo que se toquem.

Se você analisar sua tela de toque, verá quatro conectores no cabo de fita que levam a quatro tiras metálicas nas bordas da tela. Duas das tiras metálicas estão no topo e na base da tela, e se você virar a tela, verá as outras duas na segunda camada e nos lados direito e esquerdo da tela.

Quando um dedo ou stylus pressiona a camada flexível, ela se curva tocando a camada rígida, fechando o circuito e criando um interruptor (Figura 12.2).

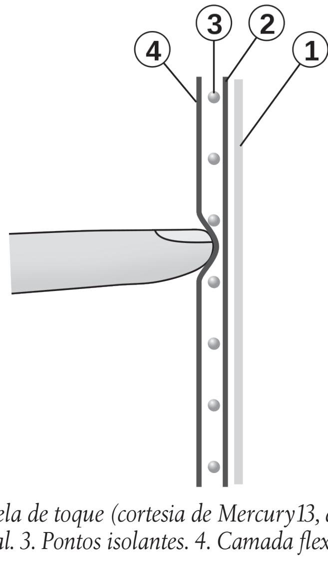


Figura 12.2 – Como funciona uma tela de toque (cortesia de Mercury13, do Wikimedia Commons). 1. Camada rígida. 2. Camada de óxido de metal. 3. Pontos isolantes. 4. Camada flexível com película de óxido de metal.

Para encontrar as coordenadas do ponto pressionado, uma tensão é aplicada ao gradiente da esquerda para a direita. Isso é feito tornando um lado o terra e o outro os 5 V. Então, uma das tiras na camada oposta é lida, utilizando uma entrada analógica para medir essa voltagem. A voltagem, quando um ponto é pressionado próximo ao lado dos 5 V, tem uma medição próxima a 5 V; da mesma forma, a voltagem, quando um ponto é pressionado próximo ao terra, tem um valor próximo a zero.

Na sequência, a voltagem é aplicada sobre a camada oposta e lida a partir da outra camada. Isso é feito em rápida sucessão centenas de vezes por segundo. Lendo rapidamente, primeiro o eixo X e depois o eixo Y, você pode obter uma voltagem para cada camada. Isso fornece uma coordenada X e Y para o ponto na tela que foi tocado. Se você tocar dois pontos na tela ao mesmo tempo, terá uma leitura igual ao ponto médio entre eles.

Há outras tecnologias utilizadas em telas de toque, mas o tipo resistivo custa pouco e sua interface com o Arduino é muito fácil, sem exigir outro tipo de circuito para que funcione adequadamente. Agora que você sabe como a tela funciona, vamos analisar o código e descobrir como medir voltagens para obter as coordenadas.

Projeto 33 – Tela de toque básica – Análise do código

O código para a leitura de uma tela de toque é, de fato, bem simples. Você inicia definindo os quatro pinos digitais que utilizará para aplicar a alimentação às camadas, e os dois pinos analógicos que serão utilizados para medir as voltagens:

```
// Conexões de alimentação
#define Left 8   // Esquerda (X1) para o pino digital 8
#define Bottom 9 // Base (Y2) para o pino digital 9
#define Right 10 // Direita (X2) para o pino digital 10
#define Top 11   // Topo (Y1) para o pino digital 11

// Conexões analógicas
#define topInput 0   // Topo (Y1) para o pino analógico 0
#define rightInput 1 // Direita (X2) para o pino analógico 1
```

Então você declara e inicializa os inteiros que armazenarão as coordenadas X e Y, ambos inicialmente definidos como 0:

```
int coordX = 0, coordY = 0;
```

Como você vai ler as coordenadas utilizando o monitor serial, no procedimento de inicialização basta iniciar a comunicação serial e definir sua taxa de transferência. Nesse caso, você utilizará uma taxa de `38400`:

```
Serial.begin(38400);
```

O loop principal do programa é composto apenas de uma instrução `if`, para determinar se a tela foi ou não tocada:

```
if (touch())  // Se a tela foi tocada, imprime as coordenadas
```

`touch()` vem a seguir. Se a tela foi tocada, você simplesmente imprime as coordenadas X e Y no monitor serial com um espaço entre elas, utilizando comandos `Serial.print`:

```
Serial.print(coordX);
Serial.print("  ");
Serial.println(coordY);
```

Depois de imprimir as coordenadas, você espera um quarto de segundo para que elas possam ser lidas, caso você mantenha seu dedo pressionado sobre a tela:

```
delay(250);
```

Na sequência, temos a função que cuida de todo o trabalho pesado. Ela retornará um valor booleano verdadeiro ou falso, por isso seu tipo é `boolean`. Você não passa nenhum parâmetro a ela, por isso a lista de parâmetros está vazia:

```
boolean touch()
```

Você declara uma variável de tipo booleano, inicializando-a como `false`. Ela armazenará um valor de `true` ou `false`, dependendo se a tela foi ou não tocada.

```
boolean touch = false;
```

Depois, você deve transmitir uma voltagem pela camada da horizontal, e ler a voltagem utilizando o pino de entrada do topo na segunda camada. Para fazê-lo, você define os pinos da direita e da esquerda como saídas, tornando o pino da esquerda `LOW` para que ele seja o terra, e o pino da direita `HIGH` para que passem por ele cinco volts:

```
pinMode(Left, OUTPUT);
digitalWrite(Left, LOW); // Define o lado esquerdo para o terra

pinMode(Right, OUTPUT);  // Define o lado direito para os +5 V
digitalWrite(Right, HIGH);
```

Os pinos do topo e da base são, então, definidos como `INPUT`, para que se tornem de alta impedância:

```
pinMode(Top, INPUT);
pinMode(Bottom, INPUT);
```

Alta impedância significa simplesmente que os pinos não serão controlados pelo circuito, e que, portanto, estarão *flutuando*; ou seja, esses pinos não serão nem `HIGH`, nem `LOW`. Você não deseja que esses pinos tenham uma voltagem passando por eles, nem que sejam terra, por isso o estado de alta impedância é perfeito, uma vez que você lerá uma voltagem analógica utilizando um desses pinos.

Na sequência, você espera um pequeno intervalo de tempo, para permitir que as alterações de estado anteriores ocorram e, em seguida, lê o valor analógico do pino de entrada do topo. Esse valor é, então, armazenado em `coordX`, fornecendo a coordenada X:

```
delay(3);
coordX = analogRead(topInput);
```

Com isso, você tem sua coordenada X. Agora, seguimos exatamente o mesmo procedimento, mas dessa vez definimos a voltagem na camada do topo até a base, e lemos o valor utilizando o pino `rightInput` da camada oposta:

```
pinMode(Bottom, OUTPUT); // Define a base para o terra
digitalWrite(Bottom, LOW);

pinMode(Top, OUTPUT);    // Defina o topo para os +5 V
digitalWrite(Top, HIGH);

pinMode(Right, INPUT);   // Esquerda e direita com impedância alta
pinMode(Left, INPUT);

delay(3);
coordY = analogRead(rightInput);
```

Você define a variável booleana `touch` como `true` apenas se os valores lidos forem maiores do que zero e menores do que mil. Isso é feito para garantir que você retorne um valor `true` apenas se as leituras estiverem dentro de valores aceitáveis:

```
If (coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}
```

Você verá que os valores variam de aproximadamente cem na escala mais baixa, até cerca de 900 no topo. Por fim, você retorna o valor de `touch`, que será `false`, se a tela não tiver sido tocada, e `true`, se isso tiver acontecido:

```
return touch;
```

Como você pode ver, a leitura de valores da tela de toque é muito simples, e permite todo tipo de utilização. Você pode colocar uma imagem ou diagrama por trás da tela, relacionando botões ou outros controles, ou sobrepor a tela a um display LCD, como em um Nintendo DS, alterando a interface do usuário sob as telas conforme necessário.

Agora avançaremos para uma demonstração simples, imprimindo um teclado que pode ser posicionado sob a tela de toque e lendo os valores apropriados, para verificar qual tecla foi pressionada.

Projeto 34 – Tela de toque com teclado

Agora você posicionará uma interface de usuário sob a tela de toque, na forma de um teclado impresso, e determinará, a partir da posição dos toques, qual de suas teclas foi pressionada. Assim que você compreender os elementos básicos desse processo, poderá substituir o teclado impresso por um exibido em um display LCD ou OLED.

A tecla pressionada será exibida em um display LCD, por isso, será necessário adicionar um componente desse tipo à lista do projeto.

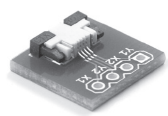
Componentes necessários

Você utilizará exatamente os mesmos componentes do projeto 33, com a adição de um display LCD 16 x 2;

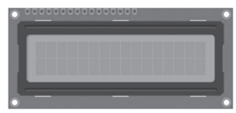
Tela de toque do Nintendo DS



Breakout para tela de toque



Display LCD 16 x 2



A outra diferença neste projeto é que você deve criar e imprimir um teclado para ser posicionado sob a tela de toque. A tela de toque usual do Nintendo DS tem 70 mm x 55 mm (2,75" x 2,16"), por isso você terá de criar um modelo com essas dimensões, utilizando um software de edição de imagens ou processador de texto, posicionando um conjunto de teclas espaçadas em um retângulo, de modo a lembrar um teclado telefônico. A figura 12.3 mostra o teclado que criei neste projeto. Sinta-se livre para utilizá-lo.

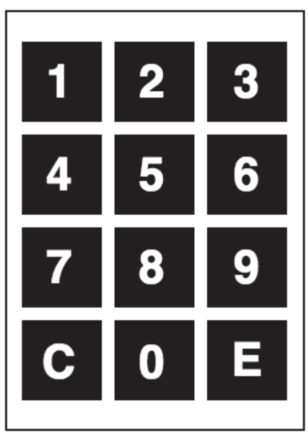


Figura 12.3 – Diagrama do teclado para o projeto 34.

Conectando os componentes

Conecte tudo como na figura 12.4.

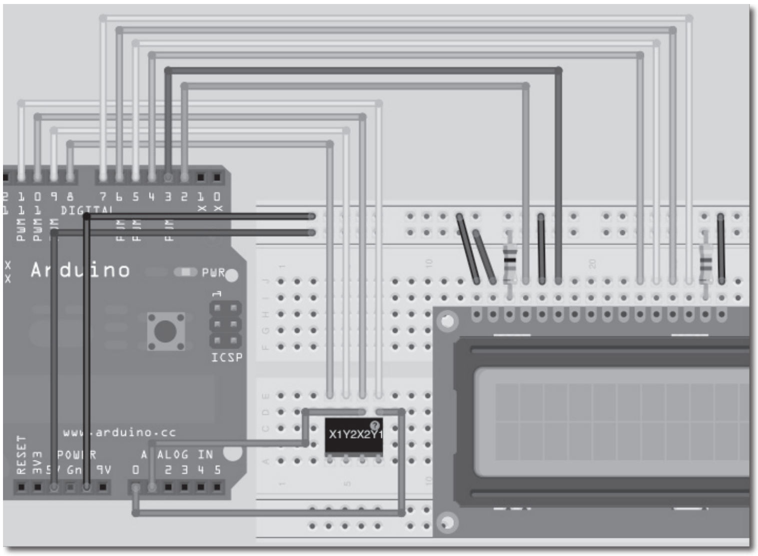


Figura 12.4 – Circuito para o Projeto 34 – Tela de toque com teclado (consulte o site da Novatec para versão colorida).

Consulte a tabela 12.2 para a pinagem do LCD.

Tabela 12.2 – Pinagem para o LCD no projeto 34

Arduino	Outro	Matriz
Digital 2		Enable
Digital 3		RS (seleção de registro)
Digital 4		DB4 (pino de dados 4)
Digital 5		DB5 (pino de dados 5)
Digital 6		DB6 (pino de dados 6)
Digital 7		DB7 (pino de dados 7)
	Terra (Gnd)	VSS (GND)
	Terra (Gnd)	R/W (leitura/escrita)
	+5 V	Vdd
	+5 V via resistor	Vo (contraste)
	+5 V via resistor	A/Vee (alimentação para o LED)
	Terra (Gnd)	Gnd para o LED

Digite o código

Digite o código da listagem 12.2.

Listagem 12.2 – Código para o projeto 34

```
// Projeto 34

#include <LiquidCrystal.h>

LiquidCrystal lcd(2, 3, 4, 5, 6, 7);  // cria um objeto lcd e atribui os pinos

// Conexões de alimentação
#define Left 8   // Esquerda (X1) para o pino digital 8
#define Bottom 9 // Base (Y2) para o pino digital 9
#define Right 10 // Direita (X2) para o pino digital 10
#define Top 11   // Topo (Y1) para o pino digital 11

// Conexões analógicas
#define topInput 0    // Topo (Y1) para o pino analógico 0
#define rightInput 1  // Direita (X2) para o pino analógico 1

int coordX = 0, coordY = 0;
char buffer[16];

void setup()
{
  lcd.begin(16, 2);    // Define o display com 16 colunas e 2 linhas
  lcd.clear();
}

void loop()
{
  if (touch())
  {
    if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
    if ((coordX>110 && coordX<300) && (coordY>410 && coordY<610)) {lcd.print("2");}
    if ((coordX>110 && coordX<300) && (coordY>640 && coordY<860)) {lcd.print("1");}
    if ((coordX>330 && coordX<470) && (coordY>170 && coordY<360)) {lcd.print("6");}
    if ((coordX>330 && coordX<470) && (coordY>410 && coordY<610)) {lcd.print("5");}
    if ((coordX>330 && coordX<470) && (coordY>640 && coordY<860)) {lcd.print("4");}
    if ((coordX>490 && coordX<710) && (coordY>170 && coordY<360)) {lcd.print("9");}
    if ((coordX>490 && coordX<710) && (coordY>410 && coordY<610)) {lcd.print("8");}
    if ((coordX>490 && coordX<710) && (coordY>640 && coordY<860)) {lcd.print("7");}
    if ((coordX>760 && coordX<940) && (coordY>170 && coordY<360)) {scrollLCD();}
    if ((coordX>760 && coordX<940) && (coordY>410 && coordY<610)) {lcd.print("0");}
    if ((coordX>760 && coordX<940) && (coordY>640 && coordY<860)) {lcd.clear();}
    delay(250);
  }
}
```

```
// retorna true se tocada, e define as coordenadas para touchX e touchY
boolean touch()
{
  boolean touch = false;

  // pega as coordenadas horizontais
  pinMode(Left, OUTPUT);
  digitalWrite(Left, LOW);   // Define o lado esquerdo para o terra

  pinMode(Right, OUTPUT);    // Define o lado direito para os +5 V
  digitalWrite(Right, HIGH);

  pinMode(Top, INPUT);     // Topo e base como alta impedância
  pinMode(Bottom, INPUT);

  delay(3);  // pequeno intervalo
  coordX = analogRead(topInput);

  // pega as coordenadas verticais
  pinMode(Bottom, OUTPUT);   // define a base para o terra
  digitalWrite(Bottom, LOW);

  pinMode(Top, OUTPUT);    // define o topo para os +5 V
  digitalWrite(Top, HIGH);

  pinMode(Right, INPUT);   // esquerda e direita como alta impedância
  pinMode(Left, INPUT);

  delay(3);  // pequeno intervalo
  coordY = analogRead(rightInput);

  // se as coordenadas lidas forem menores do que 1000 e maiores do que 0, a tela foi tocada

  if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}

    return touch;
}

void scrollLCD() {
  for (int scrollNum=0; scrollNum<16; scrollNum++) {
    lcd.scrollDisplayLeft();
    delay(100);
  }
  lcd.clear();
}
```

Digite o código e faça o upload dele para seu Arduino. Deslize o modelo do teclado, colocando-o sob a tela de toque com o cabo de fita no canto inferior direito (ao lado do E). Agora você pode pressionar as teclas na tela de toque. As teclas pressionadas serão exibidas no LCD. Quando você pressionar o C (de Clear), limpará o display. Quando pressionar E (de Enter), os números exibidos rolarão horizontalmente para a esquerda até desaparecerem.

Você já sabe como funcionam o LCD e a tela de toque, por isso não faremos uma análise do hardware neste projeto. Analisaremos apenas o código.

Projeto 34 – Tela de toque com teclado – Análise do código

Você inicia incluindo a biblioteca `LiquidCrystal` e criando um objeto `lcd`:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(2, 3, 4, 5, 6, 7); // cria um objeto lcd e atribui os pinos
```

Dessa vez, você está utilizando os pinos 2 e 3 para o RS e o Enable do LCD, e os pinos 4 e 7 para as linhas de dados. Em seguida, os pinos para as telas de toque são definidos, e as variáveis das coordenadas X e Y, inicializadas:

```
// Conexões de alimentação
#define Left 8   // Esquerda (X1) para o pino digital 8
#define Bottom 9 // Base (Y2) para o pino digital 9
#define Right 10 // Direita (X2) para o pino digital 10
#define Top 11   // Topo (Y1) para o pino digital 11

// Conexões analógicas
#define topInput 0    // Topo (Y1) para o pino analógico 0
#define rightInput 1  // Direita (X2) para o pino analógico 1

int coordX = 0, coordY = 0;
```

Na rotina de inicialização, você inicia o objeto `lcd`, atribuindo a ele 16 colunas e duas linhas. Você também limpa o display, deixando-o pronto para o uso:

```
lcd.begin(16, 2);    // Define o display com 16 colunas e 2 linhas
lcd.clear();
```

Assim como no projeto 33, no loop principal você tem uma instrução `if`, mas, dessa vez, você deve verificar se as coordenadas do toque estão dentro de um retângulo, que define os limites de cada botão. Se a coordenada estiver dentro do limite relevante de um botão, o número apropriado é exibido no LCD. Se o botão `C` for pressionado, limpamos o display; se o botão `E` for pressionado, chamamos a função `scrollLCD`.

```
if (touch()) {
  if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
  if ((coordX>110 && coordX<300) && (coordY>410 && coordY<610)) {lcd.print("2");}
  if ((coordX>110 && coordX<300) && (coordY>640 && coordY<860)) {lcd.print("1");}
  if ((coordX>330 && coordX<470) && (coordY>170 && coordY<360)) {lcd.print("6");}
  if ((coordX>330 && coordX<470) && (coordY>410 && coordY<610)) {lcd.print("5");}
  if ((coordX>330 && coordX<470) && (coordY>640 && coordY<860)) {lcd.print("4");}
  if ((coordX>490 && coordX<710) && (coordY>170 && coordY<360)) {lcd.print("9");}
  if ((coordX>490 && coordX<710) && (coordY>410 && coordY<610)) {lcd.print("8");}
  if ((coordX>490 && coordX<710) && (coordY>640 && coordY<860)) {lcd.print("7");}
  if ((coordX>760 && coordX<940) && (coordY>170 && coordY<360)) {scrollLCD();}
  if ((coordX>760 && coordX<940) && (coordY>410 && coordY<610)) {lcd.print("0");}
  if ((coordX>760 && coordX<940) && (coordY>640 && coordY<860)) {lcd.clear();}
  delay(250);
}
```

Cada instrução `if` é um conjunto de operadores condicionais e lógicos. Se você analisar a instrução do botão 3:

```
if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
```

Poderá ver que a primeira condição lógica E está verificando se a posição tocada está dentro das coordenadas `110` e `300` a partir da esquerda, e a segunda, dentro de `170` e `360` a partir do topo. Todas as condições devem ser atendidas para que o botão seja considerado pressionado, assim, operadores lógicos E (`&&`) são utilizados.

Para verificar as coordenadas de seus botões, simplesmente pressione gentilmente, utilizando um stylus, o lado esquerdo e direito do botão para obter as coordenadas de X. Então, repita esse processo para o topo e a base, obtendo as coordenadas de Y. Se você utilizar o projeto 33 para imprimir as coordenadas no monitor serial, poderá determinar as coordenadas exatas para as posições de seus botões, caso tenha de ajustar o código ou deseje criar seu próprio layout de botões.

Em seguida, temos a função `touch`; você já sabe como ela funciona. Por fim, temos a função `scrollLCD` que, como não retorna nenhum dado nem recebe parâmetros, é de tipo void:

```
void scrollLCD() {
```

Então temos um loop `for` que se repete 16 vezes, representando o número máximo de caracteres que podem ser digitados e exibidos:

```
for (int scrollNum=0; scrollNum<16; scrollNum++) {
```

Dentro do loop, você utiliza a função `scrollDisplayLeft()`, da biblioteca `LiquidCrystal`, para rolar horizontalmente os caracteres exibidos, um espaço para a esquerda. A isso se segue uma espera de cem milissegundos:

```
lcd.scrollDisplayLeft();
delay(100);
```

Fazendo isso 16 vezes, os números digitados deslizarão para a esquerda, dando a impressão de que foram inseridos no sistema. Você pode digitar suas próprias rotinas para realizar várias operações com os dados, depois que eles tiverem sido digitados.

Por fim, limpe o display, deixando-o pronto para receber novos dados, antes de encerrar a função e retornar ao loop principal:

```
lcd.clear();
```

Este projeto serviu para dar-lhe uma noção de como podemos selecionar seções de uma tela de toque, permitindo que você escolha áreas específicas para botões etc. O teclado de papel que utilizamos pode ser substituído por um display LCD Gráfico ou OLED, com botões desenhados. A vantagem disso é que menus e botões diferentes podem ser desenhados, dependendo do que o usuário selecionou. Utilizando essa técnica, você poderia criar uma elegante interface de tela de toque para seu projeto.

Agora, avançaremos para o controle de um LED RGB, deslizando um controle, na tela de toque, para escolher suas cores.

Projeto 35 – Controlador de luz com tela de toque

Neste projeto, você utilizará a tela de toque para acender e apagar uma lâmpada LED RGB e controlar a cor do LED.

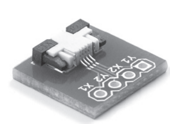
Componentes necessários

Você utilizará exatamente os mesmos componentes do projeto 33, com a adição de um LED RGB de tipo cátodo comum. Isso significa que um dos pinos é conectado ao terra (o cátodo), enquanto os outros três vão separadamente até os pinos de controle, para as voltagens de vermelho, verde e azul.

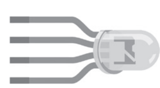
Tela de toque do Nintendo DS



Breakout para a tela de toque



LED RGB (cátodo comum)



Resistor limitador de corrente *



* se necessário

Você também deve utilizar um modelo de teclado, assim como no projeto 34. Dessa vez, serão necessárias áreas para os controles deslizantes das cores e os botões on/off. Sinta-se livre para utilizar a imagem da figura 12.5.

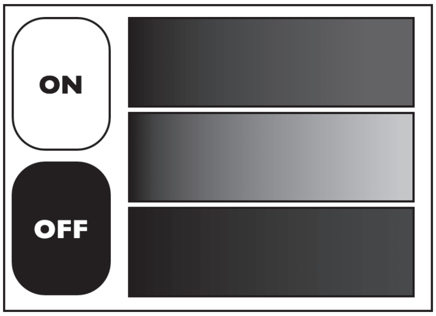


Figura 12.5 – Diagrama do teclado para o projeto 35 (consulte o site da Novatec para versão colorida).

Conectando os componentes

Conecte tudo como na figura 12.6.

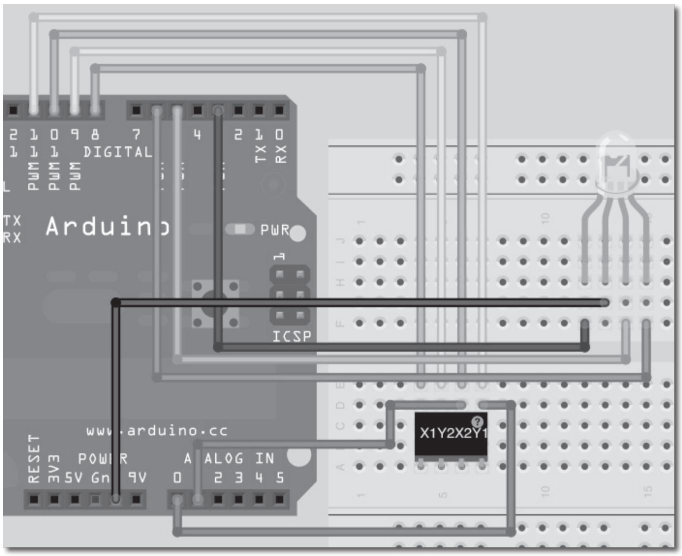


Figura 12.6 – Circuito para o Projeto 35 – Controlador de luz com tela de toque (consulte o site da Novatec para versão colorida).

O fio do terra vai para o pino do cátodo comum do LED. O pino PWM 3 vai para o ânodo vermelho, o pino PWM 5 para o verde e o pino PWM 6 para o azul. Posicione resistores limitadores de corrente nos pinos coloridos, se necessário. Utilizei um LED RGB Piranha, classificado em 5 V, por isso, em minha opinião, não tive de usar resistores. Entretanto, isso é considerado uma má prática. Utilizar LEDs sem resistores limitadores de corrente acabará por reduzir a vida útil dos componentes.

Digite o código

Digite o código da listagem 12.3.

Listagem 12.3 – Código para o projeto 35

```
// Projeto 35

// Conexões de alimentação
#define Left 8   // Esquerda (X1) para o pino digital 8
#define Bottom 9 // Base (Y2) para o pino digital 9
#define Right 10 // Direita (X2) para o pino digital 10
#define Top 11   // Topo (Y1) para o pino digital 11

// Conexões analógicas
#define topInput 0    // Topo (Y1) para o pino analógico 0
#define rightInput 1  // Direita (X2) para o pino analógico 1

// Pinos RGB
#define pinR 3
#define pinG 5
#define pinB 6

int coordX = 0, coordY = 0;
boolean ledState = true;
int red = 100, green = 100, blue = 100;

void setup() {
  pinMode(pinR, OUTPUT);
  pinMode(pinG, OUTPUT);
  pinMode(pinB, OUTPUT);
}

void loop() {
  if (touch()) {
    if ((coordX>0 && coordX<270) && (coordY>0 && coordY<460)) {ledState = true; delay(50);}
    if ((coordX>0 && coordX<270) && (coordY>510 && coordY< 880)) {ledState = false; delay(50);}
    if ((coordX>380 && coordX<930) && (coordY>0 && coordY<300)) {red= map(coordX, 380, 930, 0, 255);}
    if ((coordX>380 && coordX<930) && (coordY>350 && coordY<590)) {green=map(coordX, 380, 930, 0, 255);}
    if ((coordX>380 && coordX<930) && (coordY>640 && coordY<880)) {blue=map(coordX, 380, 930, 0, 255);}
    delay(10);
}
  if (ledState) {
    analogWrite(pinR, red);
    analogWrite(pinG, green);
    analogWrite(pinB, blue);
  }
```

```
    else {
      analogWrite(pinR, 0);
      analogWrite(pinG, 0);
      analogWrite(pinB, 0);
    }
  }

  // retorne true se tocada, e define as coordenadas para touchX e touchY
  boolean touch()
  {
    boolean touch = false;

    // pegue as coordenadas horizontais
    pinMode(Left, OUTPUT);
    digitalWrite(Left, LOW);   // Define o lado esquerdo para o terra

    pinMode(Right, OUTPUT);     // Define o lado direito para os +5 V
    digitalWrite(Right, HIGH);

    pinMode(Top, INPUT);   // Topo e base como alta impedância
    pinMode(Bottom, INPUT);

    delay(3);   // pequeno intervalo
    coordX = analogRead(topInput);

    // pegue as coordenadas verticais
    pinMode(Bottom, OUTPUT);   // define a base para o terra
    digitalWrite(Bottom, LOW);

    pinMode(Top, OUTPUT);     // define o topo para os +5 V
    digitalWrite(Top, HIGH);

    pinMode(Right, INPUT);   // esquerda e direita como alta impedância
    pinMode(Left, INPUT);

    delay(3);   // pequeno intervalo
    coordY = analogRead(rightInput);

    // se as coordenadas forem menores que 1000 e maiores que 0, a tela foi tocada

    if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}

    return touch;
  }
```

Projeto 35 – Tela de toque com controle de luz – Análise do código

Os `#defines` iniciais são os mesmos dos projetos 33 e 34, com a adição de um conjunto de `#defines` para os três pinos PWM, utilizados para controlar os componentes R, G e B do LED RGB:

```
// Pinos RGB
#define pinR 3
#define pinG 5
#define pinB 6
```

Você adiciona um valor booleano, `ledState`, definindo-o como `true`. Esse valor armazenará o estado dos LEDs, por exemplo, `true` = aceso, `false` = apagado.

```
boolean ledState = true;
```

Declaramos um conjunto de três inteiros, inicializando-os com um valor de `100` cada:

```
int red = 100, green = 100, blue = 100;
```

Esses três inteiros armazenarão separadamente os valores das cores do LED. Esses valores serão definidos pelos valores PWM emitidos dos pinos 3, 5 e 6.

Na rotina principal de inicialização, os três pinos de LED que você definiu são todos definidos como saída:

```
pinMode(pinR, OUTPUT);
pinMode(pinG, OUTPUT);
pinMode(pinB, OUTPUT);
```

No loop principal você tem, uma vez mais, uma instrução `if` para verificar se o valor retornado por `touch()` é verdadeiro. Dentro dela, temos mais instruções `if` para decidir quais partes da tela de toque foram pressionadas. As duas primeiras definem as bordas dos botões `ON` e `OFF` e alteram o `ledState` para `true`, se um toque é detectado dentro da área do botão `ON`, e para `false`, se o toque ocorre dentro da área do botão `OFF`. Um pequeno intervalo é incluído logo em seguida, para impedir leituras em falso dos botões.

```
if ((coordX>0 && coordX<270) && (coordY>0 && coordY<460)) {ledState = true; delay(50);}
if ((coordX>0 && coordX<270) && (coordY>510 && coordY< 880)) {ledState = false; delay(50);}
```

A seguir, você verifica se um toque foi detectado, utilizando as bordas das áreas do controle deslizante para as cores vermelho, verde e azul. Se um toque foi detectado, então o valor nos inteiros `red`, `green` e `blue` é alterado para corresponder à parte do controle tocada.

```
if ((coordX>380 && coordX<930) && (coordY>0 && coordY<300)) {red=map(coordX, 380, 930, 0, 255);}
if ((coordX>380 && coordX<930) && (coordY>350 && coordY<590)) {green=map(coordX, 380, 930, 0, 255);}
if ((coordX>380 && coordX<930) && (coordY>640 && coordY<880)) {blue=map(coordX, 380, 930, 0, 255);}
```

Isso pode ser feito utilizando-se uma função `map()`, que aceita cinco parâmetros. O primeiro é a variável que você está verificando, seguida pelos valores, mínimo e máximo, da variável (valores além desses limites são ignorados). Os dois parâmetros finais são os valores, mínimo e máximo, aos quais você deseja mapear os valores iniciais. Em outras palavras, você pega as coordenadas X, dentro da área do controlador deslizante, e mapeia esses valores para que se estendam de 0, no limite esquerdo, a 255,

no limite direito. Ao deslizar seu dedo da esquerda para a direita, você consegue que o componente de cor relevante mude de 0, na menor intensidade, que é apagado, a 255, no brilho máximo.

Por fim, você tem outra instrução `if`, para definir os valores PWM dos pinos R, G e B com os valores apropriados, armazenados em `red`, `green` e `blue`, mas apenas se `ledState` for `true`. Uma instrução `else` define os valores PWM todos como `0`, ou desligados, caso `ledState` seja `false`.

```
if (ledState) {
  analogWrite(pinR, red);
  analogWrite(pinG, green);
  analogWrite(pinB, blue);
}
else {
  analogWrite(pinR, 0);
  analogWrite(pinG, 0);
  analogWrite(pinB, 0);
}
```

O restante do programa corresponde à função `touch()`, a qual já abordamos.

Resumo

O projeto 35 apresentou o conceito de botões e controles deslizantes para controlar uma tela de toque. Uma vez mais, o uso de um display GLCD ou OLED resultaria em um grau de controle ainda maior sobre o sistema de iluminação. O projeto 35 poderia, com relativa facilidade, ser estendido para controlar correntes que regulariam a iluminação RGB em uma casa, substituindo interruptores padrão de luz por displays coloridos OLED e telas de toque que permitiriam um controle de luminosidade versátil.

O capítulo 12 mostrou como é fácil interfacear telas de toque com o Arduino. Com apenas um pequeno e simples programa, acompanhado de uma tela de toque e um Arduino, pode-se oferecer grande flexibilidade ao controle de usuário. Acoplada a displays gráficos, uma tela de toque se torna uma ferramenta muito útil para controle de sistemas.

Assuntos e conceitos abordados no capítulo 12:

- como utilizar uma placa breakout para facilitar a interface com conectores não padronizados;
- como funciona uma tela de toque resistiva;

- o ciclo correto de medição de alimentação e voltagem para obter as coordenadas X e Y;
- o significado de *alta impedância*;
- o fato de que telas de toque podem ser sobrepostas a displays gráficos, criando botões interativos;
- como definir uma área de botão utilizando coordenadas e operadores lógicos E;
- como áreas da tela de toque podem ser delimitadas formando botões ou controles deslizantes.

CAPÍTULO 13

Sensores de temperatura

Os dois projetos neste capítulo demonstrarão como conectar sensores de temperatura analógicos e digitais a um Arduino, e como obter suas leituras. Sensores de temperatura são muito utilizados em projetos do Arduino, em aplicações que vão desde a criação de estações meteorológicas até a produção de cerveja e projetos de balões de alta altitude. Você analisará dois sensores, o sensor analógico LM335 e o sensor digital DS18B20.

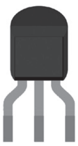
Projeto 36 – Sensor serial de temperatura

Este projeto utiliza o sensor analógico de temperatura LM335. Esse sensor é parte da série LM135 da National Semiconductors. Seu alcance é de -40°C a +100°C (-40°F a +212°F), o que o torna ideal para uma estação meteorológica, por exemplo.

Componentes necessários

O circuito e o código deste projeto foram criados para um sensor LM335, mas você pode substituir facilmente esse sensor por um LM135 ou LM235, se desejar. Caso o faça, será necessário ajustar seu código de acordo com o sensor relevante. O potenciômetro trimmer de 5 kΩ pode ser substituído por um potenciômetro rotativo padrão, de valor semelhante. Qualquer regulador de valor ou potenciômetro com valor entre 5 kΩ e 10 kΩ será suficiente.

Sensor de temperatura LM335



Potenciômetro trim de 5 kΩ



Resistor de 2,2 kΩ



Um potenciômetro trimmer, ou trim pot, é simplesmente um pequeno potenciômetro projetado para ajustar, ou regular, parte de um circuito e, uma vez calibrado, você pode esquecer que ele existe.

Conectando os componentes

Conecte tudo como na figura 13.1.

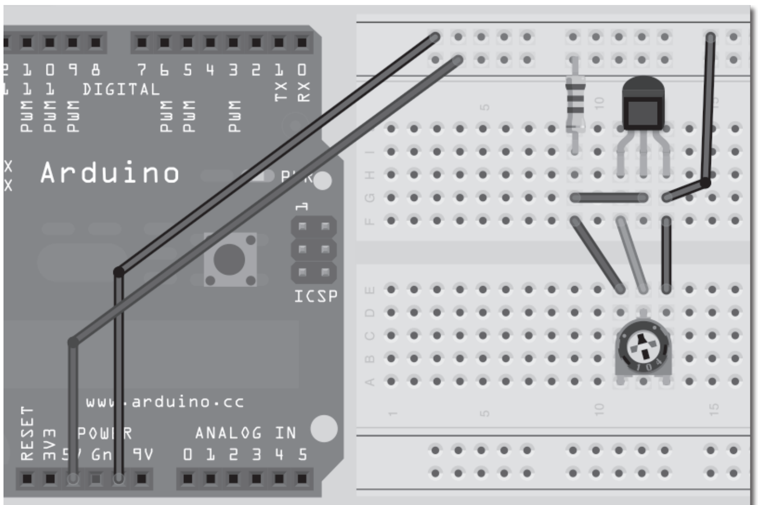


Figura 13.1 – Circuito para o Projeto 36 – Sensor serial de temperatura (consulte o site da Novatec para versão colorida).

Com o lado plano do sensor de temperatura LM335 virado para você, o terminal da esquerda será o pino de ajuste, que vai para o pino do centro do potenciômetro; o terminal do meio será o pino de alimentação, e o terminal direito, o do terra. Consulte a figura 13.2, para analisar o diagrama do datasheet da National Semiconductors.

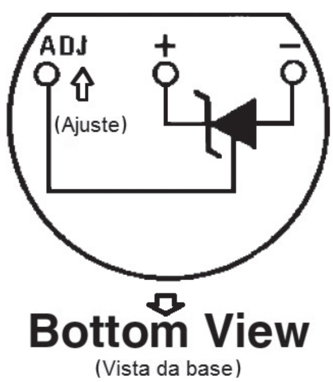


Figura 13.2 – Diagrama de pinos para o sensor de temperatura LM335.

Digite o código

Digite o código da listagem 13.1.

Listagem 13.1 – Código para o projeto 36

```
// Projeto 36

#define sensorPin 0

float Celsius, Fahrenheit, Kelvin;
int sensorValue;

void setup() {
  Serial.begin(9600);
  Serial.println("Initialising.....");
}

void loop() {

  GetTemp();
  Serial.print("Celsius: ");
  Serial.println(Celsius);
  Serial.print("Fahrenheit: ");
  Serial.println(Fahrenheit);
  Serial.println();

  delay(2000);
}

void GetTemp() {
  sensorValue = analogRead(sensorPin);              // lê o sensor
  Kelvin = (((float(sensorValue) / 1023) * 5) * 100); // converte para kelvin
  Celsius = Kelvin - 273.15;                        // converte para Celsius
  Fahrenheit = (Celsius * 1.8) +32;                 // converte para Fahrenheit
}
```

Digite o código e faça seu upload para o Arduino. Assim que o código estiver sendo executado, abra o monitor serial e certifique-se de que sua taxa de transferência esteja definida como 9.600. Você verá a temperatura tanto em Fahrenheit quanto em Celsius. Talvez a temperatura lhe pareça incorreta. É aqui que o trimmer entra em cena; você deve primeiro calibrar seu sensor. A forma mais fácil de fazê-lo é utilizando um pouco de gelo. Peque um cubo de gelo e coloque-o dentro de um saco plástico fino. Como alternativa, você pode colocar o sensor dentro de um tubo termorretrátil, com uma pequena sobreposição ao final do sensor. Quando o sensor for aquecido, se tornará à prova d'água e poderá ser mantido em contato direto com um bloco de gelo. Por isso, vá em frente e segure o cubo de gelo encostado em seu sensor por cerca de 30 segundos para que sua temperatura desça até 0°C (ou 32°F). Agora, ajuste seu trimmer ou potenciômetro, até que a leitura no monitor serial mostre a temperatura correta. Pronto, seu sensor agora está calibrado.

Você pode remover a parte do trimmer do circuito, e o circuito continuará funcionando normalmente. Entretanto, a temperatura será um valor aproximado, com limite de erro de 1°C. A forma como o sensor funciona não é importante (além de ser bem complicada), por isso analisarei apenas o código para este projeto. Se você quiser aprender mais sobre o funcionamento desse tipo de sensor, leia "The Art of Electronics", de Horowitz e Hill. Esse livro é muitas vezes chamado de "Bíblia da Eletrônica".

Projeto 36 – Sensor serial de temperatura – Análise do código

O código para este projeto é curto e simples. Você inicia definindo um pino para o sensor. Nesse caso, estamos utilizando o pino analógico 0.

```
#define sensorPin 0
```

Então, são necessárias algumas variáveis para armazenar as temperaturas em Celsius, Fahrenheit e kelvin. Como você deseja valores precisos, deve utilizar variáveis de tipo `float`.

```
float Celsius, Fahrenheit, Kelvin;
```

Depois, você cria um inteiro para armazenar o valor lido do pino analógico:

```
int sensorValue;
```

O loop de inicialização inicia a comunicação serial, a uma taxa de 9.600:

```
Serial.begin(9600);
```

Em seguida, você exibe `"Initialising....."`, para mostrar que o programa está prestes a iniciar:

```
Serial.println("Initialising.....");
```

No loop principal do programa, você chama a função `GetTemp()`, que lê a temperatura do sensor e a converte para Celsius e Fahrenheit. Depois, ela imprime as temperaturas na janela do monitor serial.

```
GetTemp();
Serial.print("Celsius: ");
Serial.println(Celsius);
Serial.print("Fahrenheit: ");
Serial.println(Fahrenheit);
Serial.println();
```

Agora, você cria a função `GetTemp()`:

```
void GetTemp()
```

Primeiramente, lemos o sensor e o valor armazenados em `sensorValue`:

```
sensorValue = analogRead(sensorPin); // lê o sensor
```

A saída do sensor está em kelvin, e cada 10 mV representam um kelvin. A medição em kelvin inicia em zero grau K, quando a temperatura é zero absoluto, temperatura mais baixa possível no universo. Assim, com um valor de zero absoluto, o sensor emitirá 0 V. De acordo com o datasheet, você pode calibrar o sensor verificando se sua voltagem é de 2,98 V, quando teremos uma temperatura de 25°C. Para converter uma temperatura de kelvin para Celsius, basta subtrair 273,15 do primeiro valor. Assim, 25°C representam, em kelvin, 298,15. Se cada grau é 10 mV, então basta mover a vírgula decimal duas casas para a esquerda, e obtemos a voltagem nessa temperatura; nesse caso, 2,98 V.

Assim, para chegar à temperatura em kelvin, você lê o valor do sensor, que varia de 0 a 1023, divide esse número por `1023` e multiplica o resultado por `5`. Isso mapeará o intervalo de 0 V a 5 V. Como cada K representa 10 mV, você multiplica esse resultado por 100, para chegar ao valor correspondente em kelvin.

```
Kelvin = (((float(sensorValue) / 1023) * 5) * 100);    // converte para kelvin
```

O valor do sensor é um inteiro, por isso o convertemos para ponto flutuante, como uma forma de garantir que o resultado também seja um valor de ponto flutuante.

Agora que você tem sua leitura em kelvin, é fácil convertê-la para Celsius e Fahrenheit. Para converter para Celsius, subtraia 273,15 da temperatura em kelvin:

```
Celsius = Kelvin - 273.15;    // converte para Celsius
```

Para converter para Fahrenheit, multiplique o valor em Celsius por 1,8 e adicione 32:

```
Fahrenheit = (Celsius * 1.8) +32;    // converte para Fahrenheit
```

A série LM135 de sensores é muito interessante, e permite que os sensores sejam facilmente calibrados, garantindo sempre uma leitura precisa. Também custam pouco, por isso você pode comprar vários deles e realizar leituras de áreas distintas de sua casa, ou de temperaturas internas e externas em um projeto de balão de alta altitude.

Outros sensores analógicos também podem ser utilizados. Pode acontecer de o terceiro pino em alguns sensores, o pino de ajuste no LM335, ser o pino de saída da temperatura. Nesse caso, você terá de utilizar esse terceiro pino para ler a temperatura, em vez do pino de alimentação. A calibração desses sensores pode ser feita facilmente com o software.

A seguir, veremos um sensor digital de temperatura, cuja opção mais popular é o DS18B20, da Dallas Semiconductor (Maxim).

Projeto 37 – Sensor digital de temperatura 1-Wire

Agora você verá como funciona o sensor digital de temperatura DS18B20. Tais sensores enviam a temperatura como um fluxo de dados seriais, transmitidos por uma única

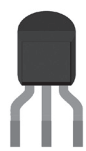
linha de transmissão – isso explica por que o protocolo é chamado de 1-Wire (1 Fio). Cada sensor também tem um número serial único, permitindo que você consulte sensores diferentes utilizando seu número de ID. Como resultado, você pode conectar muitos sensores na mesma linha de dados. Isso faz com que sensores desse tipo sejam muito populares em projetos que envolvem o Arduino, uma vez que podemos encadear uma quantidade praticamente ilimitada de sensores, conectados a apenas um único pino do Arduino. O intervalo de temperatura verificado é de -55°C a +125°C.

Você utilizará dois sensores neste projeto, para demonstrar não apenas como conectar e utilizar esse tipo de sensor, mas, também, como encadear dois ou mais sensores.

Componentes necessários

Serão necessários dois sensores no formato TO-92 (de três pinos, para que possam ser facilmente inseridos em uma protoboard, ou soldados a uma placa de circuito impresso). Alguns estão marcados como DS18B20+, o que significa que não usam chumbo.

2 sensores de temperatura DS18B20



Resistor de 4,7 kΩ



Conectando os componentes

Conecte tudo como na figura 13.3.

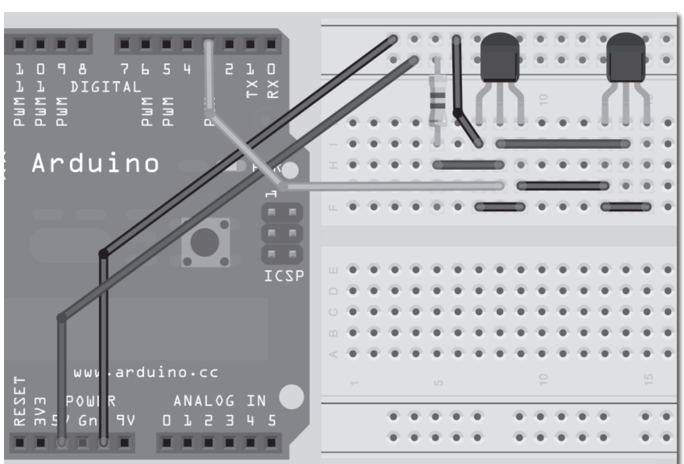


Figura 13.3 – Circuito para o Projeto 37 – Sensor digital de temperatura 1-Wire (consulte o site da Novatec para versão colorida).

Tratarei do código em duas partes. A primeira encontrará os endereços dos dois sensores. Assim que você souber esses endereços, avançará para a parte 2, na qual eles serão utilizados para obter as temperaturas diretamente dos sensores.

Digite o código

Antes que você digite o código, deve fazer o download e instalar duas bibliotecas. A primeira é a biblioteca `OneWire`. Faça seu download em *www.pjrc.com/teensy/td_libs_OneWire.html* e descompacte-a. A biblioteca `OneWire` foi escrita originalmente por Jim Stundt, tendo melhorias implementadas por Robin James, Paul Stoffregen e Tom Pollard. Essa biblioteca também pode ser utilizada para se comunicar com qualquer dispositivo 1-Wire. Coloque-a na pasta "libraries" de sua instalação do Arduino.

Depois, faça o download e instale a biblioteca `DallasTemperature`, encontrada no endereço *http://milesburton.com/index.php?title=Dallas_Temperature_Control_Library*, colocando-a também na pasta "libraries". Essa biblioteca é uma ramificação da biblioteca `OneWire`, desenvolvida por Miles Burton, com melhorias implementadas por Tim Newsome e James Whiddon. Este projeto tem por base o código dos exemplos que acompanham essa biblioteca.

Assim que você tiver instalado ambas as bibliotecas, reinicie seu Arduino e digite o código para o programa na listagem 13.2.

Listagem 13.2 – Código para o projeto 37 (Parte 1)

```
// Projeto 37 - Parte 1

#include <OneWire.h>
#include <DallasTemperature.h>

// Linha de dados vai para o pino digital 3
#define ONE_WIRE_BUS 3

// Prepara uma instância de oneWire para se comunicar com dispositivos OneWire (não apenas CIs
// de temperatura Maxim/Dallas)
OneWire oneWire(ONE_WIRE_BUS);

// Passa nossa referência a oneWire para DallasTemperature.
DallasTemperature sensors(&oneWire);

// arrays para armazenar os endereços dos dispositivos
DeviceAddress insideThermometer, outsideThermometer;

void setup()
{
```

```
  // inicia a porta serial
  Serial.begin(9600);

  // Inicia a biblioteca
  sensors.begin();

  // localiza os dispositivos no barramento
  Serial.print("Locating devices...");
  Serial.print("Found ");
  Serial.print(sensors.getDeviceCount(), DEC);
  Serial.println(" devices.");

  if (!sensors.getAddress(insideThermometer, 0))
    Serial.println("Unable to find address for Device 0");
  if (!sensors.getAddress(outsideThermometer, 1))
    Serial.println("Unable to find address for Device 1");

  // imprime os endereços para ambos os dispositivos
  Serial.print("Device 0 Address: ");
  printAddress(insideThermometer);
  Serial.println();

  Serial.print("Device 1 Address: ");
  printAddress(outsideThermometer);
  Serial.println();
  Serial.println();
}

// função para imprimir o endereço de um dispositivo
void printAddress(DeviceAddress deviceAddress)
{
  for (int i = 0; i < 8; i++)
  {
    // preenche o endereço com zeros, se necessário
    if (deviceAddress[i] < 16) Serial.print("0");
    Serial.print(deviceAddress[i], HEX);
  }
}

// função para imprimir a temperatura de um dispositivo
void printTemperature(DeviceAddress deviceAddress)
{
  float tempC = sensors.getTempC(deviceAddress);
  Serial.print("Temp C: ");
  Serial.print(tempC);
  Serial.print(" Temp F: ");
```

```
    Serial.print(DallasTemperature::toFahrenheit(tempC));
  }

  // função principal para imprimir informações sobre um dispositivo
  void printData(DeviceAddress deviceAddress)
  {
    Serial.print("Device Address: ");
    printAddress(deviceAddress);
    Serial.print(" ");
    printTemperature(deviceAddress);
    Serial.println();
  }

  void loop()
  {
    // chama sensors.requestTemperatures() para emitir uma solicitação de temperatura
    // global a todos os dispositivos no barramento
    Serial.print("Requesting temperatures...");
    sensors.requestTemperatures();
    Serial.println("DONE");

    // imprime as informações do dispositivo
    printData(insideThermometer);
    printData(outsideThermometer);
    Serial.println();
    delay(1000);
  }
```

Assim que tiver sido feito o upload do código, abra o monitor serial. Você verá um display semelhante a:

```
Locating devices...Found 2 devices.

Device 0 Address: 28CA90C202000088
Device 1 Address: 283B40C202000093

Requesting temperatures...DONE
Device Address: 28CA90C202000088 Temp C: 31.00 Temp F: 87.80
Device Address: 283B40C202000093 Temp C: 25.31 Temp F: 77.56
```

O programa principal lhe dá os números individuais de ID dos dois sensores DS18B20 utilizados. Você pode identificar cada sensor variando suas temperaturas. Segurei em minhas mãos o primeiro sensor por alguns segundos e, como você pode ver, sua temperatura subiu. Isso me diz que o sensor da direita tem endereço 28CA90C202000088, e o da esquerda, 283B40C202000093. Os endereços de seus sensores obviamente serão diferentes. Anote-os ou copie e cole a informação em seu editor de texto.

Agora que você sabe os números de ID dos dois dispositivos, pode avançar para a segunda parte. Digite o código da listagem 13.3.

Listagem 13.3 – Código para o projeto 37 (Parte 2)

```
// Projeto 37 – Parte 2

#include <OneWire.h>
#include <DallasTemperature.h>

// A conexão de dados está plugada no pino 3 do Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

// Prepara uma instância oneWire para se comunicar com dispositivos OneWire (não apenas com CIs
// de temperatura Maxim/Dallas)
OneWire oneWire(ONE_WIRE_BUS);

// Passa nossa referência a oneWire para DallasTemperature.
DallasTemperature sensors(&oneWire);

// arrays para armazenar os endereços de dispositivos – substitua pelos endereços de seus sensores
DeviceAddress insideThermometer = { 0x28, 0xCA, 0x90, 0xC2, 0x2, 0x00, 0x00, 0x88 };
DeviceAddress outsideThermometer = { 0x28, 0x3B, 0x40, 0xC2, 0x02, 0x00, 0x00, 0x93 };

void setup()
{
  // inicia a porta serial
  Serial.begin(9600);

  // Inicia a biblioteca
  sensors.begin();

  Serial.println("Initialising...");
  Serial.println();

// Define a resolução
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
}

// função para imprimir a temperatura de um dispositivo
void printTemperature(DeviceAddress deviceAddress)
{
  float tempC = sensors.getTempC(deviceAddress);
  Serial.print(" Temp C: ");
```

```
    Serial.print(tempC);
    Serial.print("  Temp F: ");
    Serial.println(DallasTemperature::toFahrenheit(tempC));
  }
  void loop()
  {
    // imprime as temperaturas
    Serial.print("Inside Temp:");
    printTemperature(insideThermometer);
    Serial.print("Outside Temp:");
    printTemperature(outsideThermometer);
    Serial.println();
    delay(3000);
  }
```

Substitua os endereços dos dois sensores por aqueles que você descobriu por meio do código da parte 1 e, então, faça o upload do código. Abra o monitor serial e você deverá encontrar uma leitura como:

```
Initialising...

Inside Temp: Temp C: 24.25  Temp F: 75.65
Outside Temp: Temp C: 19.50  Temp F: 67.10

Inside Temp: Temp C: 24.37  Temp F: 75.87
Outside Temp: Temp C: 19.44  Temp F: 66.99

Inside Temp: Temp C: 24.44  Temp F: 75.99
Outside Temp: Temp C: 19.37  Temp F: 66.87
```

Se você soldar o sensor externo a um longo fio duplo (solde os pinos 1 e 3 em um fio e o pino dois no segundo fio) e torná-lo à prova d'água (por exemplo, selando-o em um tubo termorretrátil), poderá expô-lo ao ambiente para coletar temperaturas externas. O outro sensor pode ser utilizado para obter temperaturas internas.

Projeto 37 – Sensor digital de temperatura 1-Wire – Análise do código

Primeiramente, as duas bibliotecas são incluídas:

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

Então, o pino digital que você utilizará para leitura dos dados dos sensores é definido:

```
#define ONE_WIRE_BUS 3
```

Depois, temos uma definição que determina a precisão necessária, em bits:

```
#define TEMPERATURE_PRECISION 12
```

A precisão pode ser definida com uma resolução de 9 a 12 bits. Isso corresponde a incrementos de 0,5°C, 0,25°C, 0,125°C e 0,0625°C, respectivamente. A resolução padrão é 12 bits. A resolução máxima de 12 bits resulta no menor incremento de temperatura, mas a um custo de velocidade. Na resolução máxima, o sensor demora 750ms para converter a temperatura. A 11 bits, demorará a metade do tempo (385ms), a 10 bits, demorará metade do tempo da resolução de 11 bits (187,5ms) e, finalmente, a 9 bits, 93,75ms. 750ms é suficientemente rápido para a maioria dos casos. Entretanto, se você tiver de, por algum motivo, realizar várias leituras de temperatura por segundo, a resolução de 9 bits resultará no tempo de conversão mais rápido.

Em seguida, você cria uma instância de um objeto `OneWire`, dando-lhe o nome de `oneWire`:

```
OneWire oneWire(ONE_WIRE_BUS);
```

Você também cria uma instância de um objeto `DallasTemperature`, dando-lhe o nome de `sensors`, e passa e ela uma referência ao objeto `oneWire`:

```
DallasTemperature sensors(&oneWire);
```

Depois, você deve criar os arrays que armazenarão os endereços dos sensores. A biblioteca `DallasTemperature` define variáveis de tipo `DeviceAddress` (que são apenas arrays de bytes de oito elementos). Criamos duas variáveis de tipo `DeviceAddress`, com os nomes `insideThermometer` e `outsideThermometer`, e atribuímos aos arrays os endereços encontrados na parte 1.

Simplesmente pegue os endereços encontrados na parte 1, divida-os em unidades menores de 2 dígitos hexadecimais cada, adicione 0x (para informar ao compilador que se trata de um endereço hexadecimal, e não de um decimal padrão), e separe cada uma utilizando vírgulas. O endereço será dividido em oito unidades de dois dígitos cada.

```
DeviceAddress insideThermometer = { 0x28, 0xCA, 0x90, 0xC2, 0x2, 0x00, 0x00, 0x88 };
DeviceAddress outsideThermometer = { 0x28, 0x3B, 0x40, 0xC2, 0x02, 0x00, 0x00, 0x93 };
```

No loop de inicialização, você inicia as comunicações seriais com uma taxa de transmissão de 9.600:

```
Serial.begin(9600);
```

Depois, iniciamos a comunicação com o objeto `sensors`, utilizando o comando `.begin()`:

```
sensors.begin();
```

Você imprime `"Initialising..."` para mostrar que o programa está iniciando, seguido por uma linha vazia:

```
Serial.println("Initialising...");
Serial.println();
```

Depois, você define a resolução de cada sensor utilizando o comando `.setResolution`, que exige dois parâmetros: o primeiro é o endereço do dispositivo, o segundo, sua resolução. Você já definiu a resolução no início do programa como 12 bits.

```
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

Em seguida, você cria uma função, `printTemperature()`, para imprimir a temperatura, tanto em graus Celsius quanto em kelvin, a partir do endereço do sensor definido em seu parâmetro individual:

```
void printTemperature(DeviceAddress deviceAddress)
```

Então, você utiliza o comando `.getTemp()` para obter a temperatura, em graus Celsius, a partir do endereço de dispositivo especificado. Armazenamos o resultado em uma variável de ponto flutuante, `tempC`.

```
float tempC = sensors.getTempC(deviceAddress);
```

Agora, você imprime essa temperatura:

```
Serial.print(" Temp C: ");
Serial.print(tempC);
```

Seguida pela temperatura em graus Fahrenheit:

```
Serial.print("  Temp F: ");
Serial.println(DallasTemperature::toFahrenheit(tempC));
```

Você utiliza `::` para acessar a função `toFahrenheit`, que se encontra dentro da biblioteca `DallasTemperature`. Isso converte o valor em `tempC` para Fahrenheit.

No loop principal, você simplesmente chama duas vezes a função `printTemperature()`, passando a cada iteração o endereço do sensor interno e, depois, o endereço do sensor externo, seguidos por uma espera de três segundos:

```
Serial.print("Inside Temp:");
printTemperature(insideThermometer);
Serial.print("Outside Temp:");
printTemperature(outsideThermometer);
Serial.println();
delay(3000);
```

Recomendo que você experimente os vários exemplos que acompanham a biblioteca DallasTemperature, para entender melhor suas diversas funções disponíveis. Também aconselho que você leia o datasheet do DS18B20. Esse sensor também pode ter alarmes internos ajustados para que disparem ao atingir determinadas condições de temperatura, o que pode ser útil para avaliar se as condições estão quentes ou frias demais.

O DS18B20 é um sensor muito versátil com amplo alcance sensorial, que tem a vantagem, quando comparado a um sensor analógico, de poder ser encadeado a outros sensores utilizando uma mesma linha de dados, ocupando apenas um único pino, independentemente de quantos sensores estiverem sendo utilizados.

No próximo capítulo, você verá um tipo totalmente diferente de sensor, que utiliza ondas sonoras.

Resumo

Neste capítulo, você trabalhou em dois projetos simples que mostraram como podemos conectar sensores de temperatura analógicos e digitais ao Arduino. Os projetos mostraram o básico da leitura de dados de cada sensor, e sua exibição no monitor serial. Sabendo como isso é feito, fica fácil fazer com que os dados sejam exibidos em um display de matriz de pontos LCD ou LED.

Sabendo como obter leituras de temperaturas utilizando sensores, o entusiasta do Arduino ganha acesso a uma série completamente nova de projetos. Voltaremos a falar sobre sensores de temperatura futuramente no livro, quando os colocarmos em prática no capítulo 17.

Assuntos e conceitos abordados no capítulo 13:

- como conectar um sensor analógico de temperatura a um Arduino;
- como utilizar um trimmer para calibrar um sensor da série LM135;
- como converter a voltagem do sensor para kelvin;
- como converter kelvin para Celsius, e Celsius para Fahrenheit;
- como tornar sensores à prova d'água utilizando tubos termorretráteis;
- como conectar um sensor de temperatura 1-Wire a um Arduino;
- o fato de que dispositivos 1-Wire podem ser encadeados;
- o fato de que dispositivos 1-Wire têm números de ID individuais;
- como definir a resolução de um sensor DS18B20;
- o fato de que resoluções mais elevadas significam velocidades mais lentas de conversão.

CAPÍTULO 14

Telêmetros ultrassônicos

Agora você conhecerá um tipo diferente de sensor, muito utilizado em robótica e aplicações industriais. O telêmetro ultrassônico é projetado para detectar a distância de um objeto, refletindo nele um pulso sonoro ultrassônico e verificando o tempo que demora até que o pulso retorne. Você utilizará um popular telêmetro ultrassônico, da série Maxbotix LV-MaxSonar de sensores, mas os conceitos que aprenderá neste capítulo poderão ser aplicados a qualquer outro tipo de telêmetro ultrassônico. Você verá, primeiro, como conectar o sensor ao Arduino e, depois, como utilizá-lo.

Projeto 38 – Telêmetro ultrassônico simples

O telêmetro ultrassônico LV-MaxSonar vem nos modelos EZ1, EZ2, EZ3 e EZ4. Todos têm o mesmo alcance, mas apresentam ângulos de feixe progressivamente mais estreitos, permitindo que você utilize o sensor mais adequado à sua aplicação específica. Utilizei um EZ3 na criação deste capítulo, mas você pode escolher qualquer um dos modelos.

Componentes necessários

LV-MaxSonar EZ3*



Capacitor eletrolítico de 100 µF



Resistor de 100 Ω



* ou qualquer um da série LV (imagem cortesia da Sparkfun)

Conectando os componentes

Conecte tudo como mostra a figura 14.1.

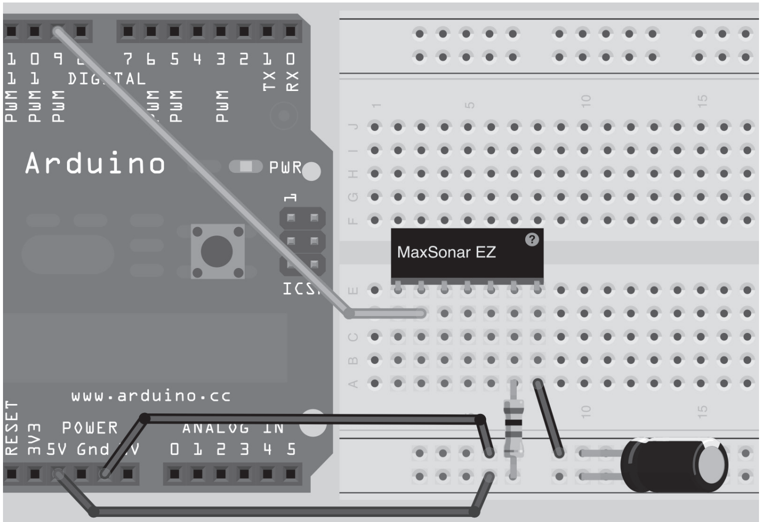


Figura 14.1 – Circuito para o Projeto 38 – Telêmetro ultrassônico simples (consulte o site da Novatec para versão colorida).

Como o Fritzing (software utilizado para criar os diagramas de protoboard deste livro) não tem um LV-MaxSonar em sua biblioteca de componentes, utilizei no diagrama um componente de tipo "mystery part" como substituto. Conecte os +5 V e o terra aos barramentos de alimentação na protoboard. Posicione um capacitor eletrolítico de 100 µF nos barramentos de alimentação, assegurando que o terminal mais longo esteja conectado aos +5 V e o mais curto (que apresenta uma faixa branca e um sinal de menos) ao barramento do terra. Então, conecte um fio jumper entre o terra e o pino Gnd no sensor. É essencial que você acerte a polaridade, uma vez que o capacitor pode explodir caso seja conectado de forma errada. Então, conecte um resistor de 100 Ω entre o barramento de +5 V e o pino de +5 V no sensor. Por fim, conecte um fio entre o pino PW no sensor e o pino digital 9.

Digite o código

Depois de verificar sua fiação, digite o código da listagem 14.1 e faça o upload para seu Arduino.

Listagem 14.1 – Código para o projeto 38

```
// Projeto 38

#define sensorPin 9

long pwmRange, inch, cm;

void setup() {
```

```
    // Inicia as comunicações seriais
    Serial.begin(115200);
    pinMode(sensorPin, INPUT);
}

void loop() {
pwmRange = pulseIn(sensorPin, HIGH);

    // 147 µS por polegada, de acordo com o datasheet
    inch = pwmRange / 147;
    // converte polegadas para cm
    cm = inch * 2.54;

    Serial.print(inch);
    Serial.print(" inches    ");
    Serial.print(cm);
    Serial.println(" cm");
}
```

Assim que você tiver feito o upload de seu código, desligue o Arduino por um segundo. Então, certifique-se de que seu sensor ultrassônico esteja imóvel e apontando para algo estático. Uma das melhores opções é colocá-lo sobre uma mesa plana e apontá-lo para o teto. Assegure-se de que não haja nada próximo ao sensor, ao religar o Arduino. Quando o dispositivo é ligado, ele executa uma rotina de calibração para o primeiro ciclo de leitura. Certifique-se de que não haja nada se movendo em seu feixe enquanto isso ocorre, do contrário você receberá leituras imprecisas. Essa informação será utilizada para determinar a distância dos objetos na linha de visão do sensor. Meça a distância entre o sensor e o teto. Essa distância será exibida (aproximadamente) no monitor serial. Se a distância não estiver correta, desligue o Arduino e ligue-o novamente, permitindo que o dispositivo calibre sem obstáculos. Movimentando o próprio sensor ou sua mão sobre ele, você verá a distância para o objeto colocado em seu alcance exibida no monitor serial.

Projeto 38 – Telêmetro ultrassônico simples – Análise do código

Novamente, você tem um exemplo de código curto e simples, na utilização desse sensor. Primeiramente, você inicia definindo o pino que utilizará para detectar o pulso. Você está utilizando o pino digital 9:

```
#define sensorPin 9
```

Então, declaramos três variáveis de tipo `long`:

```
long pwmRange, inch, cm;
```

Elas serão utilizadas para armazenar o alcance lido do sensor, o alcance convertido em polegadas e depois o alcance convertido em centímetros, respectivamente.

Na rotina de inicialização, você simplesmente inicia as comunicações seriais com uma taxa de transmissão de 115.200 e define o pino do sensor como entrada:

```
Serial.begin(115200);
pinMode(sensorPin, INPUT);
```

No loop principal, você inicia lendo o pulso do pino do sensor, armazenando-o em `pwmRange`:

```
pwmRange = pulseIn(sensorPin, HIGH);
```

Para isso, você utiliza um novo comando, `pulseIn`, perfeito para esse caso, uma vez que é projetado para medir em microssegundos o comprimento de um pulso em um pino. O pino PW do sensor envia um sinal `HIGH` quando o pulso ultrassônico é enviado a partir do dispositivo, e um sinal `LOW` quando esse pulso é recebido de volta. O tempo de oscilação do pino entre `HIGH` e `LOW` fornecerá a distância, depois de convertido. O comando `pulseIn` requer dois parâmetros. O primeiro é o pino que você deseja escutar, e o segundo é um valor `HIGH` ou `LOW`, para definir em qual estado o comando `pulseIn` iniciará a medição do pulso. Em seu caso, você define esse parâmetro como `HIGH`, de modo que tão logo o pino do sensor tenha um sinal `HIGH`, o comando `pulseIn` iniciará a medição; assim que o sinal for `LOW`, ele interromperá a medição e retornará o tempo em microssegundos.

De acordo com o datasheet para a série LV-MaxSonar de sensores, o dispositivo detectará distâncias de 0 a 254 polegadas (6,45 metros), mostrando distâncias menores que seis polegadas como 6 polegadas. Cada 147 µS (microssegundo) é igual a uma polegada. Assim, para converter o valor retornado do comando `pulseIn` para polegadas, você tem simplesmente de dividi-lo por 147. Depois, esse valor é armazenado em `inch`.

```
inch = pwmRange / 147;
```

Na sequência, esse valor é multiplicado por 2,54 para apresentar a distância em centímetros:

```
cm = inch * 2.54;
```

Por fim, os valores em polegadas e centímetros são impressos no monitor serial:

```
Serial.print(inch);
Serial.print(" inches    ");
Serial.print(cm);
Serial.println(" cm");
```

Projeto 38 – Telêmetro ultrassônico simples – Análise do hardware

O novo componente apresentado neste projeto é o telêmetro ultrassônico, dispositivo que utiliza ultrassom (som de frequência muito elevada, acima do limite da audição humana). No caso do MaxSonar, ele envia um pulso a 42 kHz. O ser humano tem um

limite máximo de audição de cerca de 20 kHz; o sensor está bem acima desse valor. Um pulso de som ultrassônico é emitido pelo dispositivo a partir de um transdutor e recebido de volta pelo mesmo transdutor, depois de refletir em um objeto. Calculando o tempo que leva para que o pulso retorne, você pode calcular a distância até o objeto refletido (Figura 14.2). Ondas sonoras viajam à velocidade do som, que em ar seco e a 20°C (68°F) é de 343 metros por segundo, ou de 1.125 pés por segundo. De posse desses dados, você pode calcular o tempo, em microssegundos, que demora até que a onda sonora retorne ao sensor. De fato, o datasheet nos diz que o tempo de retorno de um pulso que percorre uma polegada é de 147 µS. Assim, tomando o tempo em microssegundos e dividindo-o por 147, temos a distância em polegadas, que pode ser convertida para centímetros, se necessário.

Esse princípio é também conhecido como SONAR (Sound Navigation and Ranging, ou navegação e determinação de distância pelo som), e é utilizado em submarinos para detectar a distância de outras embarcações marítimas ou a proximidade de objetos. Também é utilizado por morcegos para detectar suas presas.

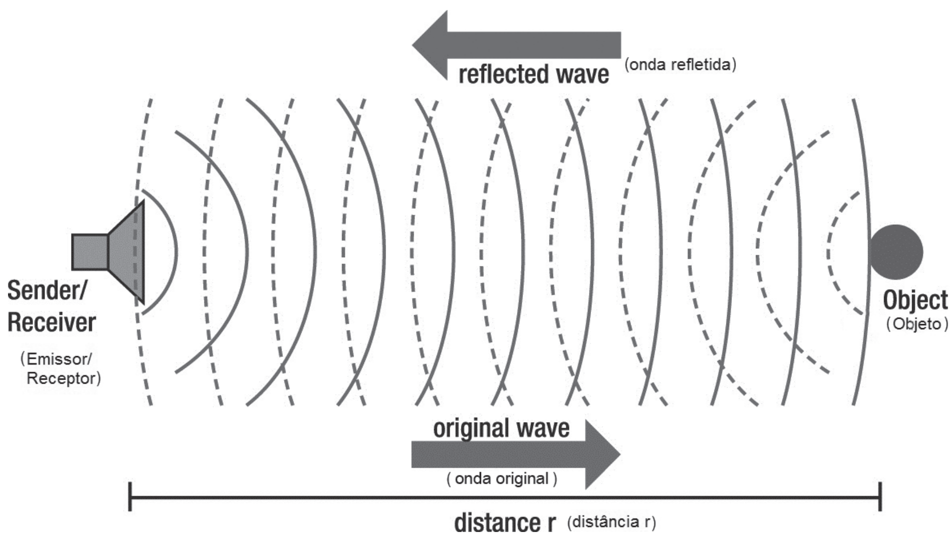


Figura 14.2 – Princípio do sonar, ou medição de distância com radar (imagem por Georg Wiora).

Dispositivos MaxSonar têm três formas de ler os dados do sensor. Uma é a entrada analógica, a segunda é uma entrada PWM e a final é uma interface serial. A entrada PWM é, provavelmente, a mais fácil de ser utilizada, e tem os dados mais confiáveis, portanto foi a que escolhemos. Sinta-se livre para pesquisar e utilizar os outros dois pinos se preferir, mesmo que não haja benefício real em fazê-lo, a menos que você necessite especificamente de um fluxo de dados analógico ou serial.

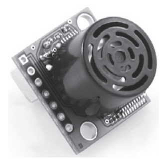
Agora que sabe como o sensor funciona, vamos utilizá-lo na prática para criar uma fita métrica ultrassônica, ou um display de distância.

Projeto 39 – Display ultrassônico de distância

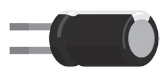
Agora você utilizará o sensor ultrassônico para criar um display de distância de precisão adequada. Você utilizará o CI controlador MAX7219, que vimos no capítulo 7, para exibir a distância medida. Todavia, em vez de uma matriz de pontos, você utilizará o recurso para o qual o MAX7219 foi projetado: displays LED de sete segmentos.

Componentes necessários

LV-MaxSonar EZ3*



Capacitor eletrolítico de 100 µF



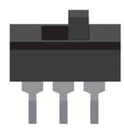
2 resistores de 100 Ω



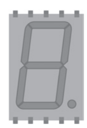
Resistor de 10 kΩ



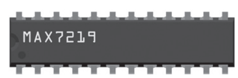
Chave seletora



5 displays LED de 7 segmentos (cátodo comum)



CI controlador de LEDs MAX7219



*ou qualquer um da série LV (imagem cortesia da Sparkfun)

A chave seletora deve ser do tipo polo único com duas posições (*single pole, double throw,* ou SPDT). Tais dispositivos apresentam uma chave deslizante que permanece em uma de duas posições possíveis. Você utilizará uma dessas posições para alternar o display entre polegadas e centímetros. Os displays LED de sete segmentos devem ser do tipo cátodo comum. Certifique-se de obter o datasheet do tipo de display adquirido, para que você possa verificar como conectá-lo corretamente, uma vez que pode haver diferenças quanto ao circuito que apresentarei.

Conectando os componentes

Conecte tudo como mostra a figura 14.3.

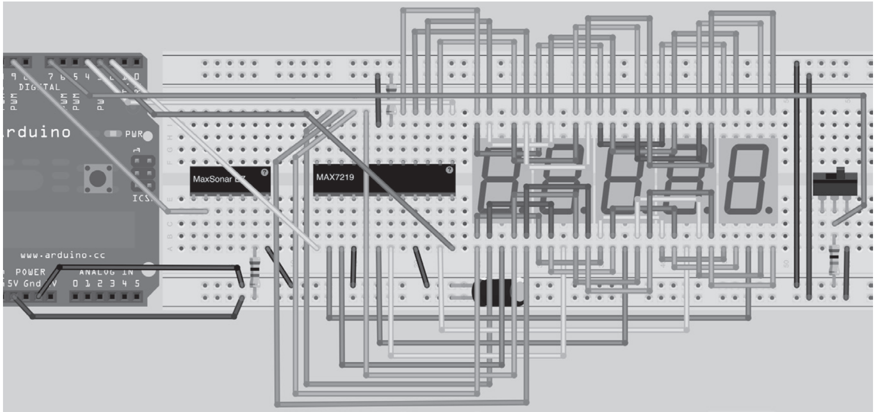


Figura 14.3 – Circuito para o Projeto 39 – Display de distância ultrassônico (consulte o site da Novatec para versão colorida).

O circuito é bem complexo, por isso forneço uma tabela de pinos para o Arduino (Tabela 14.1), o MAX7219 e o display de segmentos, de forma que você consiga corresponder ao diagrama. Os displays que utilizei tinham o código 5101AB, mas qualquer display de sete segmentos de tipo cátodo comum será suficiente. Certifique-se de que os pinos estejam no topo e na base do display, e não em suas laterais; do contrário, você não será capaz de inseri-los em uma protoboard.

Tabela 14.1 – Pinagem necessária para o projeto 39

Arduino	MaxSonar	MAX7219	Display	Outro
Pino digital 2		Pino 1 (DIN)		
Pino digital 3		Pino 12 (LOAD)		
Pino digital 4		Pino 13 (CLK)		
Pino digital 7				Chave
Pino digital 9	PW	Pino 4 (terra)		Terra (Gnd)
		Pino 9 (terra)		Terra (Gnd)
		Pino 18 (ISET)		Terra via resistor de 10 kΩ
		Pino 19 (VDD)		+5 V
		Pino 2 (DIG 0)	Terra (Gnd) no Display 0	
		Pino 11 (DIG 1)	Terra no Display 1	
		Pino 6 (DIG 2)	Terra no Display 2	
		Pino 7 (DIG 3)	Terra no Display 3	
		Pino 3 (DIG 4)	Terra no Display 4	

Arduino	MaxSonar	MAX7219	Display	Outro
		Pino 14	SEG A	
		Pino 16	SEG B	
		Pino 20	SEG C	
		Pino 23	SEG D	
		Pino 21	SEG E	
		Pino 15	SEG F	
		Pino 17	SEG G	
		Pino 22	SEG DP	

Depois de conectar o MAX7219 aos pinos SEG, de A a G, e DP do primeiro display de sete segmentos (aquele mais próximo do chip; figura 14.4), conecte os pinos SEG do primeiro display ao segundo, e depois os do segundo ao terceiro, e assim por diante. Todos os pinos SEG estão agrupados em cada display, com os pinos do terra separados e indo para os pinos DIG relevantes do MAX7219. Não se esqueça de ler o datasheet para seu display de sete segmentos, uma vez que seus pinos podem ser diferentes dos meus.

O MaxSonar está conectado da mesma forma que antes, exceto que, dessa vez, o pino PW está ligado ao pino digital 9, e não ao 3. Por fim, o pino digital 7 vai para a chave seletora.

Note que você pode ter de utilizar uma fonte de alimentação externa para este projeto, caso perceba que o circuito apresenta um comportamento errático — ele pode puxar muita energia da porta USB.

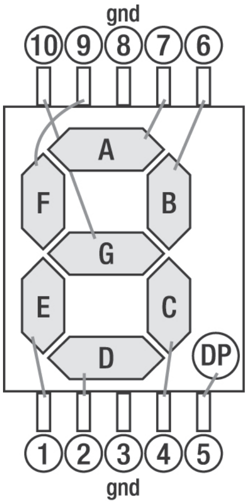


Figura 14.4 – Display de sete segmentos de cátodo comum típico, com a atribuição dos pinos (imagem cortesia de Jan Piet Mens).

Digite o código

Assim que você tiver verificado sua fiação, ligue o Arduino e digite o código da listagem 14.2, fazendo seu upload para o Arduino. Certifique-se de que `LedControl.h` esteja em sua pasta libraries (consulte o capítulo 7 para instruções).

Listagem 14.2 – Código para o projeto 39

```
// Projeto 39
#include "LedControl.h"
#define sensorPin 9
#define switchPin 7
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0
float pwmRange, averageReading, inch, cm;
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);
void setup() {
  // Acorda o MAX7219
  lc.shutdown(0,false);
  // Define-o com brilho médio
  lc.setIntensity(0,8);
  // limpa o display
  lc.clearDisplay(0);
  pinMode(sensorPin, INPUT);
  pinMode(switchPin, INPUT);
}
void loop() {
  averageReading = 0;
  for (int i = 0; i<samples; i++) {
    pwmRange = pulseIn(sensorPin, HIGH);
    averageReading += pwmRange;
  }
  averageReading /= samples;
  // 147 µS por polegada, de acordo com o datasheet
  inch = averageReading / 147;
  // converte polegadas para centímetros
  cm = inch * 2.54;
  if (digitalRead(switchPin)) {
    displayDigit(inch);
  }
  else {
```

```
    displayDigit(cm);
  }
}
void displayDigit(float value) {
  int number = value*100;
  lc.setDigit(0,4,number/10000,false);          // dígito das centenas
  lc.setDigit(0,3,(number%10000)/1000,false);  // dígito das dezenas
  lc.setDigit(0,2,(number%1000)/100,true);     // primeiro dígito com DP ligado
  lc.setDigit(0,1,(number%100)/10,false);      // dígito dos décimos
  lc.setDigit(0,0,number%10,false);            // dígito dos centésimos
}
```

Projeto 39 – Display ultrassônico de distância – Análise do código

O projeto inicia incluindo a biblioteca `LedControl.h`:

```
#include "LedControl.h"
```

Depois você define os pinos necessários para o sensor e o chip MAX7219:

```
#define sensorPin 9
#define switchPin 7
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
```

As leituras do sensor são reguladas utilizando um algoritmo simples de execução de médias, por isso você tem de definir quantas amostras devem ser tomadas para calculá-lo:

```
#define samples 5.0
```

Esse número será utilizado futuramente com `floats`, por isso, para evitar erros, nós o definimos como `5.0`, em vez de `5`, garantindo que seja forçado como um `float`, e não um `int`.

Na sequência, as variáveis de ponto flutuante são declaradas como no projeto 38, mas com a adição de `averageReading`, que será utilizada futuramente no programa:

```
float pwmRange, averageReading, inch, cm;
```

Você cria um objeto `LedControl` e define os pinos utilizados e o número de chips:

```
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);
```

Assim como no projeto 21, você se assegura de que o display esteja habilitado, define sua intensidade como média e limpa o display, deixando-o pronto para ser utilizado:

```
lc.shutdown(0,false);
lc.setIntensity(0,8);
lc.clearDisplay(0);
```

Os pinos para o sensor e a chave são ambos definidos como `INPUT`:

```
pinMode(sensorPin, INPUT);
pinMode(switchPin, INPUT);
```

Então, alcançamos o loop principal. Primeiro, definimos a variável `averageReading` como 0:

```
averageReading = 0;
```

Depois, executamos um loop `for` para coletar as amostras do sensor. O valor do sensor é transmitido a `pwmRange`, como antes, mas depois é adicionado a `averageReading` cada vez que o loop é executado. O loop `for` iterará o número de vezes definido em `samples`, no início do programa.

```
for (int i = 0; i<samples; i++) {
  pwmRange = pulseIn(sensorPin, HIGH);
  averageReading += pwmRange;
}
```

Então, você pega o valor em `averageReading` e o divide pelo número em `samples`. Em seu caso, o número foi definido como `5`, assim, cinco amostras serão tomadas, adicionadas a `averageReading` (que inicialmente é zero) e divididas por cinco para obter uma leitura média. Isso garante que você tenha uma leitura mais precisa, removendo ruídos ou alterações na medição que possam ocorrer em razão de mudanças de temperatura ou na pressão do ar.

```
averageReading /= samples;
```

Como antes, a medição do pulso é convertida em polegadas e centímetros:

```
inch = averageReading / 147;
cm = inch * 2.54;
```

Na sequência, você utiliza uma instrução `if` para verificar se a chave seletora está em `HIGH` ou em `LOW`. No caso de `HIGH`, a função `displayDigit()` (que explicaremos em breve) é executada, e o valor em polegadas é transmitido. Se a chave está em `LOW`, a instrução `else` executa a função, mas, dessa vez, utilizando centímetros:

```
if (digitalRead(switchPin)) {
  displayDigit(inch);
}
else {
  displayDigit(cm);
}
```

A instrução `if-else` assegura que um valor em polegadas ou centímetros seja exibido, dependendo da posição da chave seletora.

Por fim, você define a função `displayDigit()`, que simplesmente imprime o número transmitido no display LED de sete segmentos. Um número de ponto flutuante é transmitido à função como parâmetro. Ele estará em polegadas ou centímetros.

```
void displayDigit(float value) {
```

O número passado a essa função é um número de ponto flutuante, e terá dígitos depois da vírgula decimal. Você está interessado apenas nos dois primeiros dígitos depois da vírgula decimal, por isso ele é multiplicado por cem, para movimentar esses dois dígitos duas casas para a esquerda:

```
int number = value*100;
```

Isso ocorre porque você utilizará o operador de módulo, %, que requer números inteiros. Por isso, você deve converter o número de ponto flutuante para um inteiro. Multiplicando-o por cem, você garante que os dois dígitos depois da vírgula decimal sejam preservados, mas que todo o mais seja perdido. Agora você tem o número original, mas sem a vírgula decimal. Isso não é importante, uma vez que você sabe que há dois dígitos depois da vírgula decimal.

Em seguida, você deve pegar esse número e exibi-lo, um dígito de cada vez, nos displays de sete segmentos. Cada dígito é exibido utilizando o comando `setDigit`, que requer quatro parâmetros. Veja um exemplo:

```
setDigit(int addr, int digit, byte value, boolean dp);
```

`addr` é o endereço do chip MAX7219. Você tem apenas um único chip, por isso esse valor é 0. Se um segundo chip fosse adicionado, seu endereço seria 1, e assim por diante. `digit` é o índice do display de sete segmentos que estamos controlando. Em seu caso, o display mais à direita é o dígito 0, o display à sua esquerda é o dígito 1, e assim por diante. `value` é o dígito em si, de 0 a 9, que você deseja exibir no display LED de sete segmentos. Por fim, um valor booleano de `false` ou `true` decide se o ponto decimal desse display deve estar aceso ou apagado.

Assim, utilizando o comando `setDigit`, você pega o valor armazenado no inteiro `number` e efetua nele operações de divisão e módulo, para obter cada dígito separadamente e exibi-los no LED.

```
lc.setDigit(0,4,number/10000,false);         // dígito das centenas
lc.setDigit(0,3,(number%10000)/1000,false);  // dígito das dezenas
lc.setDigit(0,2,(number%1000)/100,true);     // primeiro dígito com DP ligado
lc.setDigit(0,1,(number%100)/10,false);      // dígito dos décimos
lc.setDigit(0,0,number%10,false);            // dígito dos centésimos
```

O dígito 2 tem sua vírgula decimal ativada, uma vez que você deseja dois dígitos depois da vírgula decimal, por isso a flag DP é `true`.

Você pode ver como isso é feito com o exemplo a seguir. Digamos que o número a ser exibido fosse 543,21. Lembre-se de que o número é multiplicado por cem, por isso você tem 54.321. Para o dígito 0, você pega o número e efetua nele uma operação de módulo 10. Isso lhe deixa com o primeiro dígito (o da direita), que é 1.

```
543.21 * 100 = 54321
54321 % 10 = 1
```

Lembre-se de que o operador de módulo, `%`, divide um inteiro pelo número depois dele, retornando o resto da divisão. `54.321` dividido por `10` seria `5432,1`, e o resto seria `1`. Isso lhe dá o primeiro dígito (o da direita) a ser exibido.

O segundo dígito (a coluna das dezenas) é obtido com uma operação de módulo 100, e depois dividido por dez, para que tenhamos o segundo dígito

```
54321 % 100 = 21
21 / 10 = 2 (lembre-se de que isso é aritmética de inteiros, por isso, qualquer número depois
da vírgula decimal será perdido)
```

E assim por diante...

Se você seguir os cálculos utilizando 543,21 como seu número original, verá que o conjunto de operações de módulo e divisão deixa você com cada dígito individual do número original. A adição do ponto decimal no dígito 2 (terceiro da direita) assegura que o número seja exibido com dois dígitos depois da vírgula decimal.

Ao terminar este projeto, você terá uma fita métrica ultrassônica com resolução de um centésimo de polegada ou de centímetro. Saiba que os resultados podem não ser exatos, uma vez que a velocidade das ondas sonoras pode depender de condições de temperatura ou da pressão atmosférica. Da mesma forma, ondas sonoras são refletidas de formas diferentes por superfícies distintas. Uma superfície perfeitamente plana, perpendicular ao plano do sensor, refletirá o som adequadamente, fornecendo as leituras mais precisas. Uma superfície com irregularidades, que absorva o som ou que esteja em ângulo com o sensor, resultará em leituras imprecisas. Teste superfícies diferentes e compare as leituras com uma fita métrica real.

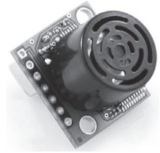
A seguir, vamos utilizar o sensor ultrassônico para algo diferente.

Projeto 40 – Alarme ultrassônico

Agora, você complementará o circuito do projeto anterior, transformando-o em um sistema de alarme.

Componentes necessários

LV-MaxSonar EZ3*



Capacitor eletrolítico de 100 µF



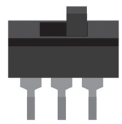
2 resistores de 100 Ω



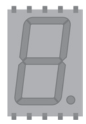
Resistor de 10 kΩ



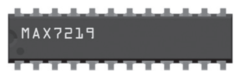
Chave seletora



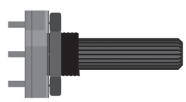
5 displays LED de 7 segmentos (cátodo comum)



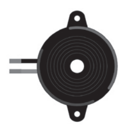
CI controlador de LEDs MAX7219



Potenciômetro de 5 a 10 kΩ



Receptor acústico Piezo ou alto-falante de 8 Ω



*ou qualquer um da série LV (imagem cortesia da Sparkfun)

Conectando os componentes

Conecte tudo como na figura 14.5.

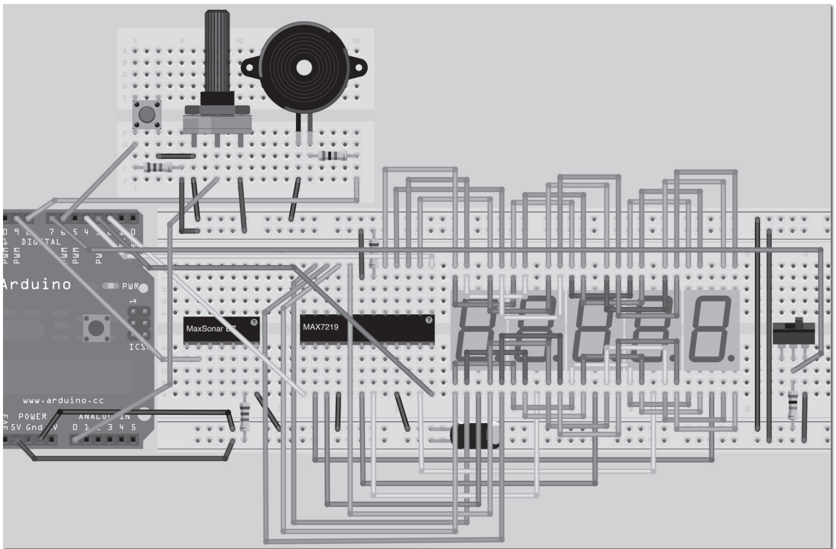


Figura 14.5 – Circuito para o Projeto 40 – Alarme ultrassônico (consulte o site da Novatec para versão colorida).

O circuito é o mesmo do projeto 39, mas com a adição de um botão, um potenciômetro e um receptor acústico piezo. O botão tem os seus terminais conectados ao +5 V e ao terra. O pino de +5 V é conectado aos +5 V por meio de um resistor de 10 kΩ. Um fio vai desse mesmo pino para o pino digital 6. O potenciômetro tem o +5 V e o terra conectados aos seus pinos externos, e o pino do meio ao pino analógico 0. O alto-falante tem seu terminal negativo conectado ao terra e o terminal positivo, por meio de um resistor de 100 Ω, conectado ao pino digital 8. O potenciômetro será utilizado para ajustar o alcance do sensor de alarme, e o botão será usado para reinicializar o sistema, depois de o alarme ter sido ativado. O piezo, obviamente, cuidará da parte sonora do alarme.

Digite o código

Depois de verificar sua fiação, ligue o Arduino e faça o upload do código da listagem 14.3.

Listagem 14.3 – Código para o projeto 40

```
// Projeto 40

#include "LedControl.h"

#define sensorPin 9
#define switchPin 7
#define buttonPin 6
#define potPin 0
#define DataIn 2
```

```
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0

float pwmRange, averageReading, inch, cm, alarmRange;
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);

void setup() {

  // Acorda o MAX7219
  lc.shutdown(0,false);
  // Define-o com brilho médio
  lc.setIntensity(0,8);
  // limpa o display
  lc.clearDisplay(0);
  pinMode(sensorPin, INPUT);
  pinMode(switchPin, INPUT);
}


void loop() {
  readPot();
  averageReading = 0;
  for (int i = 0; i<samples; i++) {
  pwmRange = pulseIn(sensorPin, HIGH);
  averageReading += pwmRange;
  }

  averageReading /= samples;
  // 147 µS por polegada, de acordo com o datasheet
  inch = averageReading / 147;
  // converte polegadas para centímetros
  cm = inch * 2.54;

  if (digitalRead(switchPin)) {
    displayDigit(inch);
  }
  else {
    displayDigit(cm);
  }

  // se o intervalo atual for menos do que alarmRange, dispara o alarme
  if (inch<=alarmRange) {startAlarm();}
}
```

```
void displayDigit(float value) {
    int number = value*100;
    lc.setDigit(0,4,number/10000,false);          // dígito das centenas
    lc.setDigit(0,3,(number%10000)/1000,false);  // dígito das dezenas
    lc.setDigit(0,2,(number%1000)/100,true);     // primeiro dígito
    lc.setDigit(0,1,(number%100)/10,false);      // dígito dos décimos
    lc.setDigit(0,0,number%10,false);            // dígito dos centésimos
}

// lê o potenciômetro
float readPot()  {
  float potValue = analogRead(potPin);
  alarmRange = 254 * (potValue/1024);
  return alarmRange;
}

// dispara o alarme, até que a reinicialização seja pressionada
void startAlarm() {
  while(1) {
    for (int freq=800; freq<2500;freq++) {
      tone(8, freq);
      if (digitalRead(buttonPin)) {
        noTone(8);
        return;
      }
    }
  }
}
```

Assim que o código tiver sido digitado, faça seu upload para o Arduino e desligue o dispositivo. Ligue-o novamente, certificando-se de que o sensor seja capaz de se calibrar corretamente. Agora você pode girar o potenciômetro para ajustar o alcance do alarme. Coloque uma de suas mãos dentro do feixe e se aproxime até que o alarme dispare. A leitura, assim que o alarme for ativado, permanecerá estável, e mostrará a você a última distância medida. Esse é o alcance de seu alarme. Pressione o botão de reset para silenciar o alarme, reinicialize o sistema e continue ajustando o potenciômetro até que obtenha um alcance satisfatório. Seu alarme agora estará pronto para proteger o que estiver perto dele. Qualquer corpo que se aproxime, e adentre o alcance do sensor que você definiu, ativará o alarme até que ele seja reinicializado.

Projeto 40 – Alarme ultrassônico – Análise do código

A maioria do código é idêntica ao que explicamos no projeto 39, por isso pularemos essas seções. Primeiro carregamos a biblioteca LedControl:

```
#include "LedControl.h"
```

Então, definimos os pinos utilizados, assim como o número de chips e de amostras, como antes:

```
#define sensorPin 9
#define switchPin 7
#define buttonPin 6
#define potPin 0
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0
```

Você adiciona uma definição para o `buttonPin` e o `potPin`. As variáveis são declaradas incluindo a nova variável, `alarmRange`, que armazenará o limiar de distância depois do qual o alarme disparará, se uma pessoa se aproximar mais do que o alcance determinado:

```
float pwmRange, averageReading, inch, cm, alarmRange;
```

Você cria um objeto `LedControl`, `lc`, e define os pinos:

```
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);
```

O loop de `setup()` é o mesmo de antes, com a adição da definição de `pinMode` e `buttonPin` como `INPUT`:

```
lc.shutdown(0,false);
lc.setIntensity(0,8);
lc.clearDisplay(0);
pinMode(sensorPin, INPUT);
pinMode(switchPin, INPUT);
pinMode(buttonPin, INPUT);
```

O loop principal inicia chamando uma nova função, `readPot()`, que lê o valor do potenciômetro utilizado para ajustar o alcance do alarme (ainda falaremos disso):

```
readPot();
```

O restante do loop principal é idêntico ao do projeto 39:

```
averageReading = 0;
for (int i = 0; i<samples; i++) {
  pwmRange = pulseIn(sensorPin, HIGH);
  averageReading += pwmRange;
}

averageReading /= samples;
inch = averageReading / 147;
cm = inch * 2.54;
```

```
if (digitalRead(switchPin)) {
  displayDigit(inch);
}
else {
  displayDigit(cm);
}
```

Até que você alcança a próxima instrução `if`:

```
if (inch<=alarmRange) {startAlarm();}
```

Essa instrução simplesmente verifica se a medição atual do sensor é menor ou igual ao valor em `alarmRange`, definido pelo usuário. Se afirmativo, ela chama a função `startAlarm()`.

A função `displayDigit()` é a mesma do projeto 39:

```
void displayDigit(float value) {
  int number = value*100;
  lc.setDigit(0,4,number/10000,false);         // dígito das centenas
  lc.setDigit(0,3,(number%10000)/1000,false);  // dígito das dezenas
  lc.setDigit(0,2,(number%1000)/100,true);     // primeiro dígito
  lc.setDigit(0,1,(number%100)/10,false);      // dígito dos décimos
  lc.setDigit(0,0,number%10,false);            // dígito dos centésimos
}
```

Em seguida, temos a primeira das duas funções novas. Essa primeira função é projetada para ler o potenciômetro e converter seu valor em polegadas, definindo o alcance do alarme. A função não tem nenhum parâmetro, mas é de tipo `float`, uma vez que retornará um valor de ponto flutuante em `alarmRange`.

```
float readPot()
```

Depois, lemos o valor analógico do `potPin` e o armazenamos em `potValue`:

```
float potValue = analogRead(potPin);
```

Então, você executa um cálculo sobre esse valor para converter os valores de 0 a 1023, lidos do potenciômetro no alcance máximo e mínimo do sensor (por exemplo, de 0 a 254 polegadas).

```
alarmRange = 254 * (potValue/1024);
```

Depois, você retorna esse valor ao ponto em que a função foi chamada:

```
return alarmRange;
```

A próxima função é responsável por disparar o som do alarme. Trata-se da função `startAlarm()`:

```
void startAlarm() {
```

Na sequência, temos um loop `while`. Você encontrou loops desse tipo no capítulo 3. Esse tipo de loop será executado enquanto a instrução dos parênteses for verdadeira. O parâmetro para o loop `while` é `1`. Isso significa simplesmente que o loop será executado enquanto o valor verificado for verdadeiro. Nesse caso, verificamos um valor constante de `1`, por isso o loop será executado eternamente. Você utilizará um comando `return` para encerrar o loop.

```
while(1) {
```

Agora você tem um loop `for` que varrerá as frequências de 800 Hz a 2.500 Hz:

```
for (int freq=800; freq<2500;freq++) {
```

Você toca um tom no pino 8, onde se encontra o receptor acústico piezo, com a frequência armazenada em `freq`:

```
tone(8, freq);
```

Agora você verifica o `buttonPin`, utilizando `digitalRead` para ver se o botão foi ou não pressionado:

```
if (digitalRead(buttonPin)) {
```

Se o botão foi pressionado, o código dentro dos parênteses é executado, iniciando com um comando `noTone()` para cessar o som do alarme, e depois um `return` para encerrar a função e retornar ao loop principal do programa:

```
noTone(8);
return;
```

No próximo projeto, você manterá o mesmo circuito, mas fará o upload de um código um pouco diferente para utilizar o sensor com um propósito distinto.

Projeto 41 – Teremim ultrassônico

Para este projeto você utilizará o mesmo circuito. Mesmo não utilizando um potenciômetro, uma chave ou um botão, manterei esses componentes no circuito, para que você tenha a flexibilidade de modificar o projeto se desejar — além disso, dessa forma você pode voltar a utilizar o projeto 40 quando quiser.

Dessa vez, você utilizará um sensor para criar um teremim, que utilizará o alcance do sensor em vez do campo elétrico que um teremim verdadeiro utiliza. Se você não sabe o que é um teremim, consulte a Wikipédia. Basicamente, trata-se de um instrumento eletrônico que você toca sem contato físico, posicionando suas mãos dentro de um campo elétrico e movimentando-as. O dispositivo percebe alterações no campo e toca uma nota que se relaciona com a distância até a bobina. É difícil explicá-lo. Para entender melhor como ele funciona, você pode verificar alguns vídeos

desse instrumento em uso no YouTube. Como o circuito é o mesmo, avançaremos diretamente para o código.

Digite o código

Digite o código da listagem 14.4.

Listagem 14.4 – Código para o projeto 41

```
// Projeto 41

#define sensorPin 9

#define lowerFreq 123     // C3
#define upperFreq 2093    // C7
#define playHeight 36

float pwmRange, inch, cm, note;

void setup() {
  pinMode(sensorPin, INPUT);
}

void loop() {
  pwmRange = pulseIn(sensorPin, HIGH);

  inch = pwmRange / 147;
  // converte polegadas para centímetros
  cm = inch * 2.54;

  // mapeia o alcance de playHeight para as frequências mais altas e baixas
  note = map(inch, 0, playHeight, lowerFreq, upperFreq);
  if (inch<playHeight) {tone(8, note); }
  else {noTone(8);}
}
```

Depois de ter feito o upload do código para o Arduino, você pode colocar sua mão dentro do feixe do sensor, para que ele toque a nota mapeada à altura em que a mão se encontra. Movimente sua mão para cima e para baixo no feixe, e as notas tocadas acompanharão seu movimento. Você pode ajustar o alcance das frequências mais altas e baixas no código, se preferir.

Projeto 41 – Teremim ultrassônico – Análise do código

Este código é uma versão simplificada do código do projeto 40, com um pouco de código adicional para transformar o alcance do sensor em um tom que pode ser tocado no receptor acústico piezo ou no alto-falante. Você inicia definindo o pino do sensor como antes.

```
#define sensorPin 9
```

Depois, você tem algumas novas definições para as notas altas e baixas a serem tocadas, além de `playHeight`, em polegadas. `playHeight` é o alcance entre o sensor e o limite da posição que seu braço poderá ocupar, enquanto você toca o instrumento. Você pode ajustar esse alcance para um valor adequado ao seu caso.

```
#define lowerFreq 123     // C3
#define upperFreq 2093    // C7
#define playHeight 36
```

As variáveis são declaradas, tendo uma para `note`, que corresponde à nota que será tocada no alto-falante:

```
float pwmRange, inch, cm, note;
```

A rotina de incialização simplesmente define o pino do sensor como entrada:

```
pinMode(sensorPin, INPUT);
```

No loop principal, o código contém apenas o essencial. O valor do sensor é lido e convertido em polegadas:

```
pwmRange = pulseIn(sensorPin, HIGH);
inch = pwmRange / 147;
```

Em seguida, os valores em polegadas, de zero até o valor armazenado em `playHeight`, são mapeados às frequências mínima e máxima, definidas no início do programa:

```
note = map(inch, 0, playHeight, lowerFreq, upperFreq);
```

Você deseja que o tom seja tocado apenas quando sua mão estiver dentro do feixe, por isso verifica se o valor do sensor é menor ou igual a `playHeight`. Se afirmativo, sua mão deve estar dentro da área de reprodução, fazendo com que um tom seja tocado.

```
if (inch<=playHeight) {tone(8, note); }
```

Se sua mão não estiver dentro do feixe, ou tiver sido removida dele, o tom será interrompido:

```
else {noTone(8);}
```

Experimente outros valores para `playHeight`, `upperFreq` e `lowerFreq`, até obter o som desejado.

Resumo

Neste capítulo, você aprendeu como interfacear com um sensor ultrassônico. Também apresentei alguns usos do sensor, para fornecer-lhe uma noção de como ele pode ser utilizado em seus projetos. Sensores como esses são, muitas vezes, utilizados em projetos caseiros de robótica, para que um robô detecte se está próximo de uma parede ou de outro obstáculo. Também já utilizei esses recursos em projetos de girocópteros, para garantir que o veículo não batesse em paredes ou pessoas. Outra utilização típica desses recursos é para detectar a altura de um líquido dentro de um tanque ou uma banheira. Tenho certeza de que você conseguirá imaginar outros usos para esse tipo de sensor.

Assuntos e conceitos abordados no capítulo 14:

- como funciona um sensor ultrassônico;
- como ler a saída PW dos dispositivos MaxSonar;
- como utilizar um capacitor para regular a linha de alimentação;
- como utilizar o comando `pulseIn` para medir a largura dos pulsos;
- os vários usos potenciais de um telêmetro ultrassônico;
- como utilizar o MAX7219 para controlar displays de sete segmentos;
- como conectar um display cátodo comum de sete segmentos;
- como utilizar um algoritmo de média para uniformizar leituras de dados;
- como utilizar o comando `setDigit()` para exibir dígitos em LEDs de sete segmentos;
- como utilizar operadores de divisão e módulo para separar dígitos de um número longo;
- como realizar um loop infinito utilizando um comando `while(1)`.

CAPÍTULO 15

Leitura e escrita de dados em um cartão SD

Agora você aprenderá o básico da escrita e leitura de dados utilizando um cartão SD. Cartões SD representam um método simples e barato de armazenar dados, e um Arduino pode se comunicar com eles com relativa facilidade, utilizando sua interface SPI. Neste capítulo, você aprenderá o suficiente para que seja capaz de criar um novo arquivo, acrescentar dados a um arquivo existente, marcar a data e a hora em um arquivo e escrever dados nele. Isso permitirá que você utilize um cartão SD e um Arduino como um dispositivo de controle de dados, capaz de armazenar os dados que quiser. Assim que souber o básico, você utilizará esse conhecimento na prática para criar um registrador de dados (*data logger*) que utiliza marcações de horário (*timestamp*).

Projeto 42 – Operação simples de leitura/escrita em um cartão SD

Você necessitará de um cartão SD e de alguma forma de conectá-lo ao Arduino. A melhor opção é adquirir um módulo breakout para cartões SD/MMC, disponível em muitos fornecedores de componentes eletrônicos para projetos de hobby. Em meu caso, utilizei uma da Sparkfun.

Componentes necessários

Cartão SD e breakout*



3 resistores de 3,3 kΩ



3 resistores de 1,8 kΩ



* imagem cortesia da Sparkfun

Os resistores são utilizados para criar um divisor de tensão e baixar os níveis lógicos de 5 V para 3,3 V. (Note que uma opção mais segura seria utilizar um conversor dedicado de nível lógico, ainda que os resistores sejam mais fáceis.)

Conectando os componentes

Conecte tudo como mostra a figura 15.1.

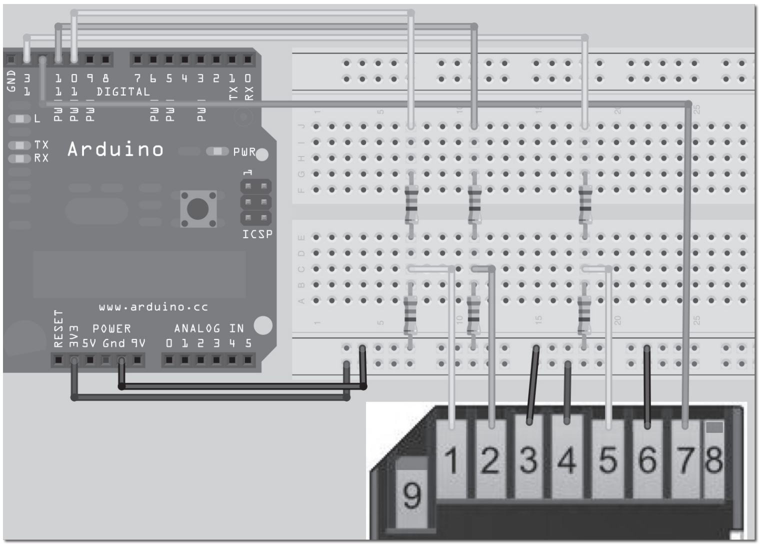


Figura 15.1 – Circuito para o Projeto 42 – Operação simples de leitura/escrita em um cartão SD (consulte o site da Novatec para versão colorida).

Consulte a tabela 15.1 para a pinagem correta. O pino digital 12 do Arduino vai direto para o pino 7 (DO) no cartão SD. Os pinos digitais 13, 11 e 10 seguem através dos resistores para baixar os níveis lógicos até 3,3 V.

Tabela 15.1 – Conexões dos pinos entre o Arduino e o cartão SD

Arduino	Cartão SD
+3,3 V	Pino 4 (VCC)
Terra (Gnd)	Pinos 3 & 6 (GND)
Pino digital 13 (SCK)	Pino 5 (CLK)
Pino digital 12 (MISO)	Pino 7 (DO)
Pino digital 11 (MOSI)	Pino 2 (DI)
Pino digital 10 (SS)	Pino 1 (CS)

Digite o código

Primeiro, você tem de instalar as bibliotecas `SdFat.h` e `SdFatUtil.h`, criadas por Bill Greiman. Atualmente, elas podem ser encontradas no endereço *http://code.google.com/p/sdfatlib/*. Faça o download da biblioteca, descompacte-a e instale a pasta `sdfat` na pasta `libraries` do seu Arduino. Assim que as bibliotecas tiverem sido instaladas e você tiver verificado sua fiação, digite o código da listagem 15.1 e faça seu upload para o Arduino.

Listagem 15.1 – Código para o projeto 42

```
// Projeto 42
// Com base nos exemplos da SD Fat, criada por Bill Greiman da sdtfatlib

#include <SdFat.h>
#include <SdFatUtil.h>

Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;

// armazena strings de erro em memória flash, para economizar a RAM
#define error(s) error_P(PSTR(s))

void error_P(const char* str) {
  PgmPrint("error: ");
  SerialPrintln_P(str);
  if (card.errorCode()) {
    PgmPrint("SD error: ");
    Serial.print(card.errorCode(), HEX);
    Serial.print(',');
    Serial.println(card.errorData(), HEX);
  }
  while(1);
}

// Escreve um retorno de carro (CR) e uma alimentação de linha (LF) no arquivo
void writeCRLF(SdFile& f) {
  f.write((uint8_t*)"\r\n", 2);
}

// Escreve um número não sinalizado no arquivo
void writeNumber(SdFile& f, uint32_t n) {
  uint8_t buf[10];
  uint8_t i = 0;
  do {
    i++;
    buf[sizeof(buf) - i] = n%10 + '0';
    n /= 10;
  } while (n);
  f.write(&buf[sizeof(buf) - i], i);
}
```

```
// Escreve uma string no arquivo
void writeString(SdFile& f, char *str) {
  uint8_t n;
  for (n = 0; str[n]; n++);
  f.write((uint8_t *)str, n);
}

void setup() {
  Serial.begin(9600);
  Serial.println();
  Serial.println("Type any character to start");
  while (!Serial.available());

  // inicializa o cartão SD em SPI_HALF_SPEED, para evitar erros de barramento com as protoboards.
  // Utilize SPI_FULL_SPEED para um melhor desempenho, caso seu cartão suporte essa opção.
  if (!card.init(SPI_HALF_SPEED)) error("card.init failed");

  // inicializa um volume FAT
  if (!volume.init(&card)) error("volume.init failed");
  // abre o diretório raiz
  if (!root.openRoot(&volume)) error("openRoot failed");

  // cria um novo arquivo
  char name[] = "TESTFILE.TXT";

  file.open(&root, name, O_CREAT | O_EXCL | O_WRITE);
  // Coloca a data e a hora atuais
  file.timestamp(2, 2010, 12, 25, 12, 34, 56);

  // escreve 10 linhas no arquivo
  for (uint8_t i = 0; i < 10; i++) {
    writeString(file, "Line: ");
    writeNumber(file, i);
    writeString(file, " Write test.");
    writeCRLF(file);
  }

  // fecha o arquivo e força uma operação de escrita de todos os dados para o cartão SD
  file.close();
  Serial.println("File Created");

  // abre um arquivo
  if (file.open(&root, name, O_READ)) {
    Serial.println(name);
  }
```

```
    else {
      error("file.open failed");
    }
    Serial.println();

    int16_t character;
    while ((character = file.read()) > 0) Serial.print((char)character);

    Serial.println("\nDone");
  }

  void loop() { }
```

Certifique-se de que seu cartão SD foi formatado recentemente em formato FAT. Execute o programa e abra o monitor serial. Será pedido que você digite um caractere e pressione `SEND`. O programa tentará escrever um arquivo no cartão SD, e imprimirá o nome desse arquivo e seu conteúdo na janela do monitor serial. Se tudo der certo, você terá um resultado como este:

```
Type any character to start
File Created
TESTFILE.TXT

    Line: 0 Write test.
    Line: 1 Write test.
    Line: 2 Write test.
    Line: 3 Write test.
    Line: 4 Write test.
    Line: 5 Write test.
    Line: 6 Write test.
    Line: 7 Write test.
    Line: 8 Write test.
    Line: 9 Write test.

    Done
```

Esteja avisado de que cartões SD, que funcionam bem em seu PC ou Mac, podem não funcionar corretamente com o Arduino. Tive de experimentar seis cartões até encontrar um que funcionasse corretamente (SD4 16 GB, da Kingston), por isso você pode ter de realizar alguns testes. Outros usuários já tiveram êxito com cartões da Sandisk.

Encerrado o programa, ejete o cartão de seu conector SD e insira-o em seu PC ou Mac. Você encontrará nele um arquivo, `TESTFILE.TXT`; se abri-lo, verá que ele contém a saída que vimos no monitor serial. Vejamos agora como funciona o código.

Projeto 42 – Operação simples de leitura/escrita em um cartão SD – Análise do código

Você inicia incluindo as duas bibliotecas do pacote da biblioteca `sdfatlib` que permitirão ao código funcionar:

```
#include <SdFat.h>
#include <SdFatUtil.h>
```

Na sequência, você tem de criar as instâncias de `Sd2Card`, `SdVolume` e `SdFile`, e nomeá-las:

```
Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;
```

O objeto `Sd2Card` oferece acesso a cartões SD e SDHC comuns. O objeto `SdVolume` oferece suporte a partições FAT16 e FAT32. O objeto `SdFile` oferece funções de acesso ao arquivo, como `open()`, `read()`, `remove()`, `write()`, `close()` e `sync()`. Esse objeto lhe fornece acesso ao diretório raiz e aos seus subdiretórios.

Em seguida, temos uma diretiva de definição para captura de erros. Você define `error(s)`, fazendo uma referência à função `error_P`:

```
#define error(s) error_P(PSTR(s))
```

Depois, você cria uma função `error_P`, cujo propósito é simplesmente imprimir mensagens de erros transmitidas a ela, assim como códigos de erros relevantes gerados. O parâmetro para a função é uma referência a uma string de caracteres. A string de caracteres é precedida por `const`. Isso é conhecido como um qualificador de variável, e serve para modificar o comportamento da variável. Nesse caso, ele qualifica a variável como somente leitura. Ela poderá ser utilizada da mesma forma que qualquer outra variável, mas seu valor não poderá ser alterado.

```
void error_P(const char* str) {
```

Em seguida, temos um comando `PGMPrint()`, parte da biblioteca `sdfatlib`, que armazena em memória flash, e imprime a string dentro de seus parênteses:

```
PgmPrint("error: ");
```

Então, temos o comando `SerialPrintln`. Uma vez mais, estamos falando de um comando da biblioteca que imprime uma string da memória flash na porta serial, seguida de um CR/LF[1] (*carriage return*, retorno de carro, e *line feed*, alimentação de linha).

```
SerialPrintln_P(str);
```

---

1 N.T.: Na computação, o carriage return (CR) é um dos caracteres de controle do código ASCII, Unicode ou EBCDIC que faz com que uma impressora ou dispositivo de saída mova a posição do cursor para a primeira posição da linha onde se encontra. Esse comando é normalmente utilizado acompanhado do line feed (avanço de linha): o retorno de carro (carriage return) antecede o avanço de linha (line feed), de maneira a indicar uma nova linha (Fonte: Wikipédia).

Depois, a função verifica se algum código de erro foi gerado, utilizando o comando `.errorCode()`. Consulte a documentação da biblioteca para uma lista de códigos de erro.

```
if (card.errorCode()) {
```

Se um erro foi gerado, o código dentro dos parênteses é executado, exibindo o código do erro e seus dados:

```
PgmPrint("SD error: ");
Serial.print(card.errorCode(), HEX);
Serial.print(',');
Serial.println(card.errorData(), HEX);
```

Por fim, se um erro ocorreu, uma função `while(1)` cria um loop infinito, para impedir que o sketch faça qualquer coisa:

```
while(1);
```

A próxima função é `writeCRLF()`, cujo objetivo é simplesmente escrever um retorno de carro (CR) e uma alimentação de linha (LF) no arquivo. O parâmetro transmitido a ela é uma referência a um arquivo.

```
void writeCRLF(SdFile& f) {
```

O código em seus parênteses utiliza o comando `write()` para escrever dois bytes no arquivo: `\r` e `\n`, códigos para operações de retorno de carro e alimentação de linha, respectivamente. O comando `write()` é o que chamamos de função sobrecarregada, ou seja, uma função definida diversas vezes para aceitar tipos de dados diferentes. Utilizamos o parâmetro `uint8_t` para dizer à função que você deseja chamar a versão da função que aceita inteiros não sinalizados de 8 bits.

```
f.write((uint8_t*)"\r\n", 2);
```

A próxima função é projetada para escrever números no arquivo; ela aceita uma referência ao arquivo e um número inteiro não sinalizado de 32 bits.

```
void writeNumber(SdFile& f, uint32_t n) {
```

Um array de comprimento 10, que armazena inteiros não sinalizados de 8 bits, é criado, assim como uma variável `i`, inicializada como `0`:

```
uint8_t buf[10];
uint8_t i = 0;
```

Em seguida, temos o loop `do-while`, cuja função é transformar engenhosamente o número inteiro em uma string, um dígito de cada vez:

```
do {
  i++;
  buf[sizeof(buf) - i] = n%10 + '0';
  n /= 10;
} while (n);
```

O loop `do-while` é algo que você ainda não viu. Ele funciona da mesma forma que um loop `while`. Entretanto, nesse caso, a condição é testada ao final do loop, e não em seu início. Enquanto o loop `while` não será executado se sua condição não for atendida, o loop `do-while` sempre executará seu código ao menos uma vez. O loop repetirá apenas se a condição for atendida.

O loop incrementa `i` e utiliza o comando `sizeof` para encontrar o tamanho do array. Essa operação retorna o número de bytes do array. Você sabe que ele tem comprimento `10`, e que `i` inicia como `1`, assim ele acessa primeiro `10 - 1`, o nono elemento do array (o último elemento). Na sequência, temos `10 - 2`, o oitavo elemento, e assim por diante, indo da direita para a esquerda. Depois, ele armazena nesse elemento o resultado de `n%10`, que representa o dígito mais à direita de qualquer número fornecido. Então, `n` é dividido por `10`, o que faz com que você remova esse dígito. Depois, o processo se repete. Fazendo isso, obtemos o dígito mais à direita do número e o armazenamos no último elemento do array; depois, eliminamos esse dígito e repetimos o processo. Ao fazê-lo, o número é dividido em dígitos individuais e armazenado no array como uma string. Para converter o número em seu equivalente ASCII, utilizamos `+ '0'` ao final.

```
buf[sizeof(buf) - i] = n%10 + '0';
```

Essa operação tem o efeito de inserir o dígito ASCII `"0"`, adicionando-o ao valor de `n %10`. Em outras palavras, se o número fosse `123`, teríamos `123 % 0 = 3`. Assim o código ASCII para `"0"`, que é 48, seria adicionado a 3, criando 51, que representa o código ASCII para `"3"`. Fazendo isso, podemos converter o dígito em seu equivalente ASCII. Assim que o loop encerra, o conteúdo do array `buff[]` é escrito no arquivo, utilizando:

```
f.write(&buf[sizeof(buf) - i], i);
```

A função recebe um ponteiro com a localização dos dados sendo escritos, seguido pelo número de bytes que devem ser escritos. Essa é mais uma versão da função sobrecarregada `write()`. Depois, temos uma função cujo propósito é escrever uma string de texto no arquivo; seus parâmetros são o arquivo e um ponteiro para a string.

```
void writeString(SdFile& f, char *str) {
```

Novamente, você utiliza simplesmente um loop `for` para analisar cada elemento do array da string, caractere por caractere, e escrevê-los no arquivo:

```
uint8_t n;
for (n = 0; str[n]; n++);
f.write((uint8_t *)str, n);
```

Em seguida, temos a rotina de inicialização, que realiza todo o trabalho pesado. Como você deseja que esse código seja executado apenas uma vez, coloque todo o código na função `setup()`, e nada em `loop()`.

Primeiramente, você inicializa a comunicação serial e, depois, pede ao usuário que digite um caractere para iniciar:

```
Serial.begin(9600);
Serial.println();
Serial.println("Type any character to start");
```

Agora o programa espera até que algo seja digitado no monitor serial, utilizando um loop `while` para ficar inativo enquanto NÃO houver nada disponível na linha serial:

```
while (!Serial.available());
```

Em seguida, temos três instruções `if`, que servem para executar a função de erro se houver algum problema na inicialização do cartão, do volume ou na abertura do diretório raiz:

```
if (!card.init(SPI_HALF_SPEED)) error("card.init failed");
if (!volume.init(&card)) error("volume.init failed");
if (!root.openRoot(&volume)) error("openRoot failed");
```

Você pode alterar a velocidade para `SPI_FULL_SPEED`, caso seu cartão suporte essa opção. Em meu caso, tive erros ao tentar a opção de velocidade máxima, por isso optei pela velocidade média. Talvez você tenha melhor sorte em seu projeto.

Agora você necessita de um nome para o novo arquivo que está prestes a criar, por isso coloque essa informação em um array `char`:

```
char name[] = "TESTFILE.TXT";
```

Em seguida, você abre um arquivo no diretório raiz:

```
file.open(&root, name, O_CREAT | O_EXCL | O_WRITE);
```

O arquivo aberto é aquele armazenado em `name`, que você acabou de inicializar como `TESTFILE.TXT`. Como esse arquivo não existe, ele será criado ao mesmo tempo. Há três flags no comando que servem para dizer-lhe o que deve ser feito: `O_CREAT`, `O_EXCL` e `O_WRITE`.

A flag `O_CREAT` diz ao comando `open` para criar o arquivo caso ele não exista. `O_EXCL` fará com que o comando falhe se `O_CREAT` também estiver definida (ou seja, o arquivo é exclusivo; assim, se ele já existe, não deve ser criado novamente). `O_WRITE` abre o arquivo para escrita. Dessa forma, em essência, esse comando fará as seguintes operações: abrirá o arquivo; criará um novo arquivo caso ele ainda não exista; assegurará que um novo arquivo não seja sobrescrito caso ele já exista; e, finalmente, abrirá o arquivo, deixando-o pronto para operações de escrita.

Em seguida, você utiliza o comando `timestamp`, para garantir que o arquivo tenha dados de data e hora corretos quando criado. O comando aceita sete parâmetros: uma flag, o ano, o mês, o dia, a hora, os minutos e os segundos.

```
file.timestamp(2, 2010, 12, 25, 12, 34, 56);
```

Nesse caso você passa a ele dados falsos, mas o ideal é que você obtenha essa informação de uma fonte como um chip RTC (Real Time Clock), ou um módulo GPS, e utilize esses dados para marcar a hora do arquivo corretamente. A flag que corresponde ao primeiro parâmetro é composta de:

```
T_ACCESS = 1
T_CREATE = 2
T_WRITE = 4
```

- `T_ACCESS` - Define a última data de acesso do arquivo;
- `T_CREATE` - Define a data e hora de criação do arquivo;
- `T_WRITE` - Define a data e hora da última operação de escrita/modificação do arquivo.

Em seu caso, você utiliza apenas o valor 2, o que significa que define a data e hora da criação do arquivo. Caso você quisesse utilizar todas as três opções ao mesmo tempo, o valor seria 7 (4 + 2 + 1).

Na sequência, um loop `for` é executado dez vezes, para escrever o número da linha e alguns dados de teste no arquivo, seguido por um retorno de carro e uma alimentação de linha. As três funções para escrita de números, strings e CRLF são chamadas nessa operação.

```
for (uint8_t i = 0; i < 10; i++) {
  writeString(file, "Line: ");
  writeNumber(file, i);
  writeString(file, " Write test.");
  writeCRLF(file);
}
```

Qualquer ação que seja executada no arquivo não será escrita até que o arquivo seja fechado. Você utiliza o comando `.close()` para isso. Dessa forma, você fecha o arquivo e escreve nele todos os dados definidos em seu código.

```
file.close();
```

Então, você deixa que o usuário saiba que o arquivo foi criado:

```
Serial.println("File Created");
```

Agora que criou um novo arquivo e escreveu dados nele, você avança para a parte do programa que abre o arquivo e lê seus dados. Para isso, utilizamos o comando `open()`, que necessita de três parâmetros: o diretório em que se encontra o arquivo, o nome deste e a flag apropriada para a operação. Você utiliza `&root` para garantir que está verificando o diretório raiz, e a flag `O_READ`, que abre um arquivo para leitura. O comando é implementado como condição de uma instrução `if`, para que você possa imprimir o nome do arquivo, caso consiga abri-lo, e executar uma rotina de erro se a operação não tiver êxito.

```
if (file.open(&root, name, O_READ)) {
  Serial.println(name);
  }
else{
  error("file.open failed");
}
Serial.println();
```

Então, você lê o arquivo, um caractere de cada vez, utilizando um loop `while` e um comando `.read()`, e imprime o resultado no monitor serial:

```
int16_t character;
while ((character = file.read()) > 0) Serial.print((char)character);
```

Por fim, se tudo der certo, você imprime "Done" (concluído):

```
  Serial.println("\nDone");
}
```

O loop principal do programa não contém nenhum código. Você deseja que o código seja executado apenas uma vez, por isso faz sentido colocar tudo na rotina de `setup()`, e nada no loop. Dessa forma, o código será executado apenas uma vez, seguido pelo loop. Como este não contém nenhum procedimento, nada acontecerá até que o Arduino seja desligado ou reinicializado:

```
void loop() { }
```

O exemplo que acabamos de analisar mostrou o método básico de criação de um arquivo, de escrita de números e strings nele, de seu fechamento e, finalmente, de sua leitura. Agora, ampliaremos esse conhecimento utilizando-o na prática, e empregaremos o cartão SD para registrar dados de um sensor.

Projeto 43 – Registrador de dados de temperatura em um cartão SD

Agora, você adicionará alguns sensores de temperatura DS18B20 ao circuito, assim como um chip RTC (Real Time Clock) DS1307. As leituras dos sensores de temperatura serão registradas no cartão SD, e você utilizará o chip RTC para ler a data e hora, de modo que todas as leituras do sensor e a modificação do arquivo tenham marcas temporais.

Componentes necessários

Cartão SD e placa breakout*



3 resistores de 3,3 kΩ



3 resistores de 1,8 kΩ



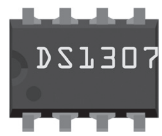
Resistor de 4,7 kΩ



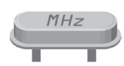
2 resistores de 1 kΩ



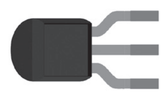
CI RTC DS1307



Cristal de relógio de 32,768 kHz e 12,5 pF



2 sensores de temperatura DS18B20



Suporte para bateria de tipo moeda**



* imagem cortesia da Sparkfun

** opcional

O suporte para bateria de tipo moeda é opcional. Uma bateria de tipo moeda como alimentação de backup permite que você retenha a hora e data no RTC, mesmo após desligar seu projeto.

Conectando os componentes

Conecte tudo como na figura 15.2.

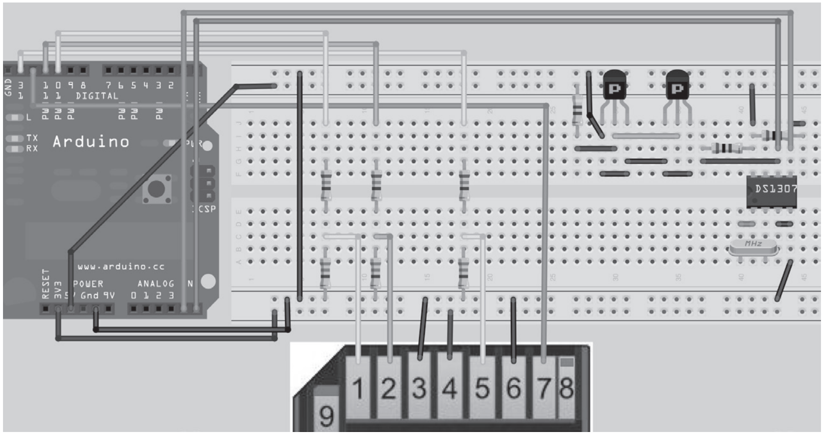


Figura 15.2 – Circuito para o Projeto 43 – Registrador de dados de temperatura em SD (consulte o site da Novatec para versão colorida).

Consulte a tabela 15.2 para a pinagem correta. Conecte o DS18B20 exatamente como no projeto 37. Caso você esteja utilizando uma bateria de backup de tipo moeda, não instale a fiação entre os pinos 3 e 4 do RTC, como no diagrama. Em vez disso, conecte o terminal positivo da bateria ao pino 3 do chip, e o negativo, ao pino 4.

Tabela 15.2 – Conexões dos pinos entre o Arduino, o cartão SD e o RTC

Arduino	Cartão SD	RTC
+5 V	-	Pino 5
+3,3 V	Pino 4 (VCC)	
Terra (Gnd)	Pinos 3 & 6 (GND)	Pinos 3 & 4
Pino digital 13 (SCK)	Pino 5 (CLK)	
Pino digital 12 (MISO)	Pino 7 (DO)	
Pino digital 11 (MOSI)	Pino 2 (DI)	
Pino digital 10 (SS)	Pino 1 (CS)	
Pino analógico 4		Pino 5
Pino analógico 5		Pino 6

Posicione os resistores de 1 kΩ entre o pino 8 e os pinos 5 e 6 do RTC.

Digite o código

Certifique-se de que as bibliotecas `OneWire.h` e `DallasTemperature.h`, utilizadas no projeto 37, estejam instaladas para este projeto. Você também utilizará a biblioteca `DS1307.h`, de Matt Joyce e D. Sjunnesson, para controlar o chip DS1307.

Listagem 15.2 – Código para o projeto 43

```
// Projeto 43
// Com base nos exemplos da SD Fat, criada por Bill Greiman da sdfatlib
// Biblioteca DS1307, de Matt Joyce, com melhorias criadas por D. Sjunnesson

#include <SdFat.h>
#include <SdFatUtil.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <WProgram.h>
#include <Wire.h>
#include <DS1307.h>    // escrita por mattt, no forum do Arduino, e modificada por D. Sjunnesson

// armazena as strings de erro em memória flash, para economizar a RAM
#define error(s) error_P(PSTR(s))
// O fio de dados está conectado ao pino 3 do Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;

// Prepara uma instância de OneWire para se comunicar com qualquer dispositivo OneWire (não
// apenas CIs de temperatura Maxim/Dallas)
OneWire oneWire(ONE_WIRE_BUS);
// Passa uma referência à instância oneWire para Dallas Temperature.
DallasTemperature sensors(&oneWire);

// arrays para armazenar os endereços do dispositivo
DeviceAddress insideThermometer = { 0x10, 0x20, 0x2C, 0xA9, 0x01, 0x08, 0x00, 0x73 };
DeviceAddress outsideThermometer = { 0x10, 0x22, 0x5B, 0xA9, 0x01, 0x08, 0x00, 0x21 };

float tempC, tempF;
int hour, minute, seconds, day, month, year;

// cria um novo nome de arquivo
char name[] = "TEMPLOG.TXT";

void error_P(const char* str) {
  PgmPrint("error: ");
  SerialPrintln_P(str);
  if (card.errorCode()) {
    PgmPrint("SD error: ");
```

```
      Serial.print(card.errorCode(), HEX);
      Serial.print(',');
      Serial.println(card.errorData(), HEX);
    }
    while(1);
}

void writeCRLF(SdFile& f) {
  f.write((uint8_t*)"\r\n", 2);
}

// Escreve um número não sinalizado no arquivo
void writeNumber(SdFile& f, uint32_t n) {
  uint8_t buf[10];
  uint8_t i = 0;
  do {
    i++;
    buf[sizeof(buf) - i] = n%10 + '0';
    n /= 10;
  } while (n);
  f.write(&buf[sizeof(buf) - i], i);
}

// Escreve uma string no arquivo
void writeString(SdFile& f, char *str) {
  uint8_t n;
  for (n = 0; str[n]; n++);
  f.write((uint8_t *)str, n);
}

void getTemperature(DeviceAddress deviceAddress)
{
  sensors.requestTemperatures();
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}

void getTimeDate() {
  hour = RTC.get(DS1307_HR,true);       // lê a hora e também atualiza todos os valores,
                                        // utilizando true
  minute = RTC.get(DS1307_MIN,false);   // lê os minutos sem atualização (false)
  seconds = RTC.get(DS1307_SEC,false);  // lê os segundos
  day = RTC.get(DS1307_DATE,false);     // lê a data
  month = RTC.get(DS1307_MTH,false);    // lê o mês
  year = RTC.get(DS1307_YR,false);      // lê o ano
}
```

```
void setup() {
  Serial.begin(9600);
  Serial.println("Type any character to start");
  while (!Serial.available());
  Serial.println();
  // Inicia a biblioteca sensors
  sensors.begin();
  Serial.println("Initialising Sensors.");

  // Define a resolução
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
  delay(100);
  // Define o tempo no RTC.
  // Comente esta seção, caso você já tenha definido o tempo e tenha uma bateria de backup
  RTC.stop();
  RTC.set(DS1307_SEC,0);    // define os segundos
  RTC.set(DS1307_MIN,15);   // define os minutos
  RTC.set(DS1307_HR,14);    // define as horas
  RTC.set(DS1307_DOW,7);    // define o dia da semana
  RTC.set(DS1307_DATE,3);   // define a data
  RTC.set(DS1307_MTH,10);   // define o mês
  RTC.set(DS1307_YR,10);    // define o ano
  RTC.start();

  Serial.println("Initialising SD Card...");

  // inicializa o cartão SD em SPI_HALF_SPEED, para evitar erros de barramento com as protoboards.
  // Utiliza SPI_FULL_SPEED para um melhor desempenho, caso seu cartão suporte essa opção.
  if (!card.init(SPI_HALF_SPEED)) error("card.init failed");

  // inicializa um volume FAT
  if (!volume.init(&card)) error("volume.init failed");

  // abre o diretório raiz
  if (!root.openRoot(&volume)) error("openRoot failed");
  Serial.println("SD Card initialised successfully.");
  Serial.println();
}
void loop() {
  Serial.println("File Opened.");
  file.open(&root, name, O_CREAT | O_APPEND | O_WRITE);
  getTimeDate();
  file.timestamp(7, year, month, day, hour, minute, seconds);
```

```
    getTemperature(insideThermometer);
    Serial.print("Inside: ");
    Serial.print(tempC);
    Serial.print(" C  ");
    Serial.print(tempF);
    Serial.println(" F");
    writeNumber(file, year);
    writeString(file, "/");
    writeNumber(file, month);
    writeString(file, "/");
    writeNumber(file, day);
    writeString(file, "  ");
    writeNumber(file, hour);
    writeString(file, ":");
    writeNumber(file, minute);
    writeString(file, ":");
    writeNumber(file, seconds);
    writeCRLF(file);
    writeString(file, "Internal Sensor: ");
    writeNumber(file, tempC);
    writeString(file, " C   ");
    writeNumber(file, tempF);
    writeString(file, " F");
    writeCRLF(file);

    getTemperature(outsideThermometer);
    Serial.print("Outside: ");
    Serial.print(tempC);
    Serial.print(" C  ");
    Serial.print(tempF);
    Serial.println(" F");
    writeString(file, "External Sensor: ");
    writeNumber(file, tempC);
    writeString(file, " C   ");
    writeNumber(file, tempF);
    writeString(file, " F");
    writeCRLF(file);
    writeCRLF(file);

    Serial.println("Data written.");
    // fecha o arquivo e força a escrita de todos os dados no cartão SD
    file.close();
    Serial.println("File Closed.");
    Serial.println();
    delay(10000);
  }
```

Abra o monitor serial, e o programa pedirá que você digite um caractere para iniciar o código. Ao fazê-lo, você terá uma saída semelhante a esta:

```
Type any character to start

Initialising Sensors.
Initialising SD Card...
SD Card initialised successfully.

File Opened.
Inside: 27.25 C  81.05 F
Outside: 15.19 C  59.34 F
Data written.
File Closed.

File Opened.
Inside: 28.31 C  82.96 F
Outside: 15.25 C  59.45 F
Data written.
File Closed.

File Opened.
Inside: 28.62 C  83.52 F
Outside: 15.31 C  59.56 F
Data written.
File Closed.
```

Se você desligar o Arduino, ejetar o cartão SD e inserir esse cartão em seu PC ou Mac, você verá um arquivo `TEMPLOG.TXT`. Abra esse arquivo em um editor de texto, e você verá leituras do sensor com marcas temporais, como estas:

```
2010/10/3  17:29:9
Internal Sensor: 27 C    81 F
External Sensor: 15 C    59 F

2010/10/3  17:29:21
Internal Sensor: 28 C    82 F
External Sensor: 15 C    59 F

2010/10/3  17:29:32
Internal Sensor: 28 C    83 F
External Sensor: 15 C    59 F
```

Vejamos como funciona o programa.

Projeto 43 – Registrador de dados de temperatura em um cartão SD – Análise do código

Como muito deste código foi abordado nos projetos 37 e 42, falaremos apenas dos novos elementos.

Primeiramente, as bibliotecas apropriadas são incluídas:

```
#include <SdFat.h>
#include <SdFatUtil.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <WProgram.h>
#include <Wire.h>
#include <DS1307.h>
```

As três últimas bibliotecas são novas e necessárias para executar o código do DS1307. As bibliotecas `WProgram.h` e `Wire.h` acompanham o programa principal do Arduino.

Depois, as definições para captura de erro e o barramento do dispositivo 1-Wire são criados:

```
#define error(s) error_P(PSTR(s))
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

Então, criamos objetos para o cartão SD:

```
Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;
```

Você também cria instâncias para o dispositivo 1-Wire e para o sensor de temperatura Dallas:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

Então, você define os endereços para os sensores DS18B20. Utilize o projeto 37, parte 1, para descobrir os endereços de seus sensores.

```
DeviceAddress insideThermometer = { 0x10, 0x20, 0x2C, 0xA9, 0x01, 0x08, 0x00, 0x73 };
DeviceAddress outsideThermometer = { 0x10, 0x22, 0x5B, 0xA9, 0x01, 0x08, 0x00, 0x21 };
```

Agora, você cria algumas variáveis que armazenarão as leituras de temperatura e de data e hora obtidas do RTC, além do array que armazenará o nome do arquivo em que você registrará os dados:

```
float tempC, tempF;
int hour, minute, seconds, day, month, year;
char name[] = "TEMPLOG.TXT";
```

Em seguida, temos as funções para captura de erro, para escrita de um CR e de um LF no arquivo, e também para as operações de escrita dos números e strings:

```
void error_P(const char* str) {
  PgmPrint("error: ");
  SerialPrintln_P(str);
  if (card.errorCode()) {
    PgmPrint("SD error: ");
    Serial.print(card.errorCode(), HEX);
    Serial.print(',');
    Serial.println(card.errorData(), HEX);
  }
  while(1);
}

void writeCRLF(SdFile& f) {
  f.write((uint8_t*)"\r\n", 2);
}

// Escreve um número não sinalizado no arquivo
void writeNumber(SdFile& f, uint32_t n) {
  uint8_t buf[10];
  uint8_t i = 0;
  do {
    i++;
    buf[sizeof(buf) - i] = n%10 + '0';
    n /= 10;
  } while (n);
  f.write(&buf[sizeof(buf) - i], i);
}

// Escreve uma string no arquivo
void writeString(SdFile& f, char *str) {
  uint8_t n;
  for (n = 0; str[n]; n++);
  f.write((uint8_t *)str, n);
}
```

Agora, você cria uma nova função para obter as temperaturas do sensor. O endereço do dispositivo é informado como parâmetro.

```
void getTemperature(DeviceAddress deviceAddress) {
  sensors.requestTemperatures();
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}
```

Em seguida, você cria outra função, `getTimeDate()`, para obter a hora e a data do chip DS1307 de relógio em tempo real.

```
void getTimeDate() {
```

Para obter os dados referentes às horas, aos minutos, aos segundos, ao dia, mês e ano do RTC, você utiliza o comando `.get()`. Primeiramente, você obtém a hora do dispositivo, armazenando-a na variável `hour`.

```
hour = RTC.get(DS1307_HR,true); // lê a hora e também atualiza todos os valores, utilizando true
```

O comando requer dois parâmetros. O primeiro é uma flag para declarar qual elemento dos dados você deseja. Seus nomes tornam óbvias suas funções. O segundo parâmetro é `false` ou `true`. Se `true`, as constantes de tempo (`DS1307_HR`, `DS1307_YR` etc.) serão atualizadas com a data e hora atuais. Se `false`, elas simplesmente lerão os dados da última atualização. Como você deseja atualizar essas constantes apenas uma vez, no início da leitura de data, coloque uma flag `true` no primeiro comando `.get()` e uma `false` no restante.

```
minute = RTC.get(DS1307_MIN,false);    // lê os minutos sem atualização (false)
seconds = RTC.get(DS1307_SEC,false);  // lê os segundos
day = RTC.get(DS1307_DATE,false);     // lê a data
month = RTC.get(DS1307_MTH,false);    // lê o mês
year = RTC.get(DS1307_YR,false);      // lê o ano
```

Depois, temos a rotina de inicialização:

```
void setup() {
Serial.begin(9600);
Serial.println("Type any character to start");
while (!Serial.available());
Serial.println();
```

Os sensores DS18B20 são inicializados e sua resolução é definida:

```
sensors.begin();
Serial.println("Initialising Sensors.");
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

Em seguida, temos a seção que define a data e a hora no RTC. Você utiliza o comando `.set()`, cuja função é, essencialmente, o inverso da apresentada pelo comando `.get()`. Antes que você defina o dispositivo, deve utilizar o comando `.stop()` para interrompê-lo. Assim que o tempo tiver sido definido, o comando `.start()` será utilizado para iniciar seu RTC com hora e data atualizadas.

```
RTC.stop();
RTC.set(DS1307_SEC,0);   // define os segundos
RTC.set(DS1307_MIN,15);  // define os minutos
RTC.set(DS1307_HR,14);   // define as horas
```

```
RTC.set(DS1307_DOW,7);   // define o dia da semana
RTC.set(DS1307_DATE,3);  // define a data
RTC.set(DS1307_MTH,10);  // define o mês
RTC.set(DS1307_YR,10);   // define o ano
RTC.start();
```

O cartão SD é inicializado:

```
Serial.println("Initialising SD Card...");

// inicializa o cartão SD em SPI_HALF_SPEED, para evitar erros de barramento com as protoboards.
// Utilize SPI_FULL_SPEED para um melhor desempenho, caso seu cartão suporte essa opção.
if (!card.init(SPI_HALF_SPEED)) error("card.init failed");

// inicializa um volume FAT
if (!volume.init(&card)) error("volume.init failed");

// abre o diretório raiz
if (!root.openRoot(&volume)) error("openRoot failed");
Serial.println("SD Card initialised successfully.");
Serial.println();
```

Em seguida, temos o loop principal. Nele, abrimos o arquivo. Dessa vez, você utiliza a flag `O_APPEND`.

```
file.open(&root, name, O_CREAT | O_APPEND | O_WRITE);
```

Depois, você chama a função `getTimeDate()`, para obter as definições de hora e data do RTC:

```
getTimeDate();
```

Esses valores são utilizados para criar a marcação de horário (*timestamp*) do arquivo:

```
file.timestamp(7, year, month, day, hour, minute, seconds);
```

Agora, você chama a função `getTemperature()`, para obter a temperatura do primeiro dispositivo:

```
getTemperature(insideThermometer);
```

As temperaturas são impressas no monitor serial, seguidas da escrita da marcação de horário, e depois as temperaturas são escritas no arquivo:

```
Serial.print("Inside: ");
Serial.print(tempC);
Serial.print(" C  ");
Serial.print(tempF);
Serial.println(" F");
writeNumber(file, year);
writeString(file, "/");
writeNumber(file, month);
```

```
writeString(file, "/");
writeNumber(file, day);
writeString(file, "  ");
writeNumber(file, hour);
writeString(file, ":");
writeNumber(file, minute);
writeString(file, ":");
writeNumber(file, seconds);
writeCRLF(file);
writeString(file, "Internal Sensor: ");
writeNumber(file, tempC);
writeString(file, " C   ");
writeNumber(file, tempF);
writeString(file, " F");
writeCRLF(file);
```

Então, você obtém a temperatura do segundo dispositivo e repete o mesmo processo (menos a seção da marcação de horário):

```
getTemperature(outsideThermometer);
Serial.print("Outside: ");
Serial.print(tempC);
Serial.print(" C  ");
Serial.print(tempF);
Serial.println(" F");
writeString(file, "External Sensor: ");
writeNumber(file, tempC);
writeString(file, " C   ");
writeNumber(file, tempF);
writeString(file, " F");
writeCRLF(file);
writeCRLF(file);
```

Por fim, fechamos o arquivo e informamos essa ação ao usuário, seguido por uma espera de dez segundos até a próxima leitura:

```
Serial.println("Data written.");
// fecha o arquivo e força a escrita de todos os dados no cartão SD
file.close();
Serial.println("File Closed.");
Serial.println();
delay(10000);
```

Agora você tem uma noção básica do que deve ser feito para escrever dados do sensor (ou de outros dispositivos) em um cartão SD. Para funcionalidades mais avançadas, leia a documentação que acompanha a biblioteca `SDfat`. Em seguida, veremos uma breve explicação sobre o chip RTC, antes de avançarmos ao próximo capítulo.

Projeto 43 – Registrador de dados de temperatura em um cartão SD – Análise do hardware

No projeto 43, você foi apresentado a um novo CI, o chip de relógio em tempo real DS1307. Esse incrível CI permite que você adicione facilmente um relógio aos seus projetos. Com a adição de um suporte de bateria de tipo moeda, você pode desconectar seu Arduino da alimentação, e o chip automaticamente passará a utilizar a bateria de backup, mantendo seus dados de data e hora atualizados, utilizando a bateria. Com um cristal de boa qualidade, o dispositivo será capaz de manter dados razoavelmente precisos. O dispositivo é capaz até mesmo de trabalhar com anos bissextos e meses com menos de 31 dias. Além disso, ele também pode ser definido para operar em modos de 24 ou 12 horas. A comunicação com o dispositivo é feita por meio de uma interface I2C, que explicarei em breve.

O chip é interessante, pois também emite uma onda quadrada no pino 7, que pode ser de 1 Hz, 4,096 kHz, 8,192 kHz ou 32,768 kHz. Assim, você também poderia utilizá-lo como um oscilador para gerar sons, ou com outros propósitos que exijam um pulso. Você poderia facilmente adicionar um LED a esse pino para indicar os segundos que passam, se definisse o pulso como 1 Hz.

A comunicação com o chip é feita utilizando um protocolo chamado I2C. Você já encontrou comunicações de tipo 1-Wire e SPI, mas esse é um protocolo novo. Para utilizá-lo, você tem de incluir a biblioteca `Wire.h` em seu código.

O protocolo I2C (ou Inter-IC), às vezes chamado TWI, ou Two Wire Interface, foi desenvolvido pela Philips Semiconductors (também conhecida como NXP) para criar um barramento bidirecional simples, utilizando apenas duas linhas de comunicação para controle entre os CIs. O protocolo utiliza apenas duas linhas: a linha serial de dados (serial data line, ou SDA) e a linha serial do clock (serial clock line, ou SCL). No Arduino, elas correspondem ao pino analógico 4 (SDA) e ao pino analógico 5 (SCL). O único outro hardware externo exigido é um resistor pull-up em cada uma das linhas do barramento. Você pode ter até 128 dispositivos I2C (também chamados de nós) conectados às mesmas duas linhas de comunicação. Alguns dispositivos I2C utilizam +5 V, outros +3,3 V, por isso você deve estar atento a essa característica quando for utilizá-los. Certifique-se de ler o datasheet e de utilizar a voltagem correta, antes de conectar os componentes de um dispositivo I2C. Se você deseja ter dois Arduinos se comunicando, o protocolo I2C será uma boa escolha para seu projeto.

O protocolo I2C é semelhante ao SPI no sentido de que há dispositivos mestre e escravo. O Arduino é o mestre, e o dispositivo I2C é o escravo. Cada dispositivo tem seu próprio endereço I2C. Um bit de dado é enviado a cada pulso do clock. A comunicação inicia quando o mestre emite uma condição START no barramento, e encerra quando ele emite uma condição STOP.

Para iniciar uma comunicação I2C em um Arduino, você utiliza o comando `Wire.begin()`. Isso inicializará a biblioteca Wire e definirá o Arduino como o dispositivo I2C mestre. Também configurará os pinos analógicos 4 e 5 como pinos I2C. Para iniciar a comunicação como um escravo (por exemplo, dois Arduinos conectados utilizando I2C), o endereço do dispositivo escravo tem de ser incluído nos parênteses. Em outras palavras, o comando:

```
Wire.begin(5);
```

Fará com que o segundo Arduino se junte ao barramento I2C como dispositivo escravo no endereço 5. Um byte pode ser recebido do dispositivo I2C, utilizando:

```
Int x = Wire. Receive();
```

Antes de fazê-lo, você deve solicitar o número de bytes, utilizando `Wire.requestFrom(address, quantity)`.

Assim, o comando:

```
Wire.requestFrom(5,10);
```

Solicitaria dez bytes do dispositivo 5. Enviar dados para o dispositivo é tão simples quanto utilizar o comando:

```
Wire.beginTransmission(5);
```

O qual define que a transmissão deve ser feita para o dispositivo de número 5. Depois, podemos utilizar:

```
Wire.send(x);
```

Para enviar um byte, ou:

```
Wire.send("Wire test.");
```

Para enviar dez bytes.

Você pode aprender mais sobre a biblioteca Wire em *www.arduino.cc/en/Reference/Wire*, e sobre o protocolo I2C na Wikipédia, ou lendo a excelente explicação desse protocolo oferecida pelos datasheets da Atmega no site da Atmel.

Resumo

No capítulo 15, você aprendeu o básico da leitura e escrita de dados em um cartão SD. Lendo a documentação que acompanha a biblioteca SDFat, há muitos outros conceitos que você pode aprender sobre o uso de um cartão SD. Com os projetos deste capítulo, apenas arranhamos a superfície do que é possível. No processo, você foi apresentado ao protocolo I2C. Também aprendeu a conectar um chip DS1307 de relógio em tempo real, que será de grande utilidade para seus futuros projetos com

relógios. Outra ótima maneira de obter um sinal preciso de tempo é por meio de um dispositivo GPS com saída serial.

Saber como ler e escrever dados em um cartão SD é conhecimento vital para criar registradores de dados, especialmente para dispositivos de alimentação remota. Muitos dos projetos de balões de alta altitude (High Altitude Balloons, ou HAB), com base no Arduino ou em chips AVR, utilizam cartões SD para registrar dados de GPS e sensores, permitindo que essas informações sejam acessadas quando o balão é recuperado.

Assuntos e conceitos abordados no capítulo 15:

- como conectar um cartão SD a um Arduino;
- como utilizar circuitos de divisores de tensão com resistores, para diminuir os níveis de voltagem de +5 V para +3,3 V;
- como utilizar a biblioteca SDFat;
- como escrever strings e números em arquivos;
- como abrir e fechar arquivos;
- como nomear arquivos;
- como criar marcações de horário (*timestamps*) em arquivos;
- como capturar erros em arquivos;
- o conceito do loop `do-while`;
- como utilizar operações de módulo e divisão para obter dígitos individuais de um número mais longo;
- como conectar um chip DS1307 de relógio em tempo real a um Arduino;
- como utilizar a biblioteca `DS1307.h` para acessar dados de data e hora;
- como utilizar uma bateria de backup no DS1307 para reter dados, caso haja uma falta de energia;
- uma introdução ao protocolo I2C.

CAPÍTULO 16

Criação de um leitor RFID

Leitores RFID (Radio Frequency Identification, identificação por radiofrequência) são muito populares hoje em dia, e se tornaram a escolha preferida para controle de entrada de pessoas em prédios comerciais, bem como em sistemas de transporte público. Pequenas etiquetas (tags) RFID são injetadas em animais para identificação, caso estes se percam. Veículos têm tags para recolhimentos de pedágios. Elas chegam até mesmo a ser utilizadas em hospitais, para marcar equipamentos descartáveis em salas de cirurgia, como uma forma de garantir que nenhum objeto estranho seja esquecido dentro dos pacientes. O custo dessa tecnologia caiu drasticamente com o passar dos anos, a ponto de os leitores, agora, poderem ser comprados por menos de dez dólares. Da mesma forma, não é difícil conectá-los ao Arduino. Como resultado, muitos projetos interessantes podem ser criados com eles.

Você utilizará o leitor ID-12 da Innovations, que não custa caro e pode ser adquirido com facilidade. Esses leitores utilizam tecnologia de 125 kHz, para ler tags e cartões a até 180 mm de distância. Há outros leitores da mesma série que oferecem um alcance maior, e você pode estender essa capacidade ainda mais, adicionando antenas externas.

Iniciaremos este capítulo conectando um desses leitores, e aprenderemos a obter dados seriais a partir dele. Depois, você criará um sistema de controle de acesso simples.

Projeto 44 – Leitor RFID simples

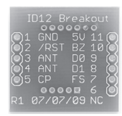
Os pinos nos leitores ID-12 têm espaçamento não regular, por isso não podem ser encaixados na protoboard. Você terá de obter uma placa breakout em um fornecedor como a Sparkfun. Os cartões e as tags podem ser adquiridos em todo tipo de fornecedor, e custam muito pouco; comprei muitas tags de estilo pingente no eBay por apenas alguns dólares. Certifique-se de que a tag, ou o cartão, seja de tecnologia de 125 kHz; do contrário, o componente não funcionará com esse leitor.

Componentes necessários

Leitor RFID ID-12



Placa breakout para o ID-12*



Resistor limitador de corrente



LED de 5 mm



Tags ou cartões de 125 kHz (ao menos 4)

* imagem cortesia da Sparkfun

Conectando os componentes

Conecte tudo como na figura 16.1.

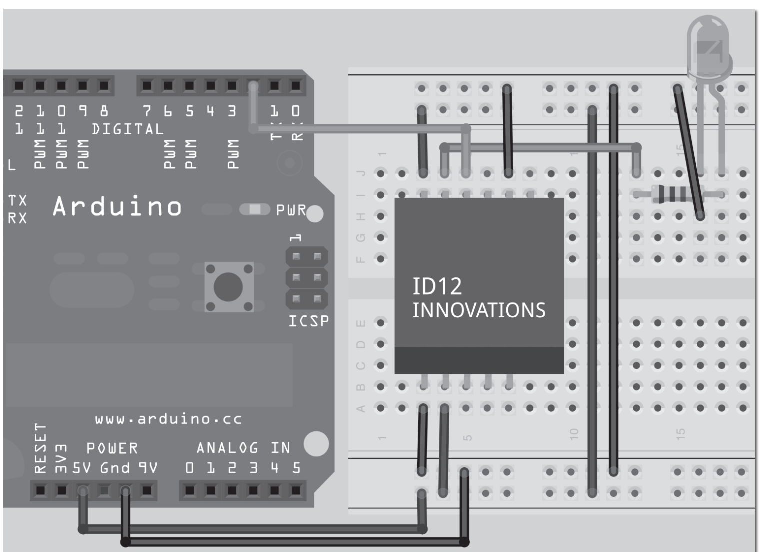


Figura 16.1 – Circuito para o Projeto 44 – Leitor RFID simples (consulte o site da Novatec para versão colorida).

Conecte um LED por meio de um resistor limitador de corrente ao pino 10 (BZ) do leitor.

Digite o código

Digite o código da listagem 16.1.

Listagem 16.1 – Código para o projeto 44

```
// Projeto 44

char val = 0; // valor lido na porta serial

void setup() {
  Serial.begin(9600);
}

void loop () {

  if(Serial.available() > 0) {
    val = Serial.read();      // lê da porta serial
    Serial.print(val, BYTE);  // e imprime no monitor
  }
}
```

Execute o código, abra o monitor serial e segure a tag ou o cartão RFID próximo ao leitor. O LED piscará para indicar que o cartão foi lido, e a janela do monitor serial mostrará um número de 12 dígitos: o ID individual do cartão. Tome nota dos IDs de suas tags; eles serão necessários para o próximo projeto.

O código não traz nada além de uma simples leitura dos dados presentes na porta serial. A essa altura, você já deve saber o suficiente para entender como funciona este programa. Portanto, pularemos a análise do código e veremos apenas o hardware.

Projeto 44 – Leitor RFID simples – Análise do hardware

A tecnologia RFID está em todos os lugares, de passes de ônibus às portas de acesso de seu escritório ou universidade. As tags ou os cartões que apresentam essa tecnologia são de muitos tipos e formatos diferentes (Figura 16.2), e podem ser tão pequenos que cientistas já chegaram até mesmo a fixá-los em formigas, para monitorar seus movimentos. Estamos falando de dispositivos simples, que não fazem nada além de transmitir um código serial individual, por ondas de rádio, ao leitor. Na maioria dos casos, os cartões ou as tags são *passivos*, o que significa que não têm bateria e necessitam da alimentação de uma fonte externa. Outras opções incluem *RFID ativo*, com sua própria fonte de alimentação, e *passivo com auxílio de bateria* (*battery assisted passive*, ou BAP), que aguarda ser ativado por uma fonte externa e, então, utiliza sua própria alimentação para transmitir os dados, resultando em um maior alcance.

O tipo de tag que você utilizará neste projeto é o passivo, sem bateria. Essas tags obtêm sua energia de um campo magnético transmitido pelo leitor. Quando a tag passa pelo campo magnético, este induz uma corrente nos fios dentro da tag. Essa corrente é utilizada para ativar um pequenino chip, que transmite o número serial da tag. O leitor, então, envia esses dados em formato serial ao PC ou ao microcontrolador conectado a ele. O formato dos dados enviados a partir do leitor ID-12 é o seguinte:

STX (02h)	DATA (10 ASCII)	CHECKSUM (2 ASCII)	CR	LF	ETX (03H)

Um byte de início da transmissão, ou STX, é enviado primeiro (02h), seguido por dez caracteres ASCII (equivalentes a cinco bytes) que compõem a representação dos dez dígitos hexadecimais individuais do número. Os dois caracteres ASCII seguintes (equivalentes a um byte) correspondem ao checksum desse número (conceito que será explicado no próximo projeto). Depois, temos um retorno de carro (CR) e uma alimentação de linha (LF), seguidos por um ETX (03h), código de término da transmissão.

Apenas os 12 dígitos ASCII serão mostrados no monitor serial, uma vez que o restante é formado por caracteres não imprimíveis. No próximo projeto, você utilizará o checksum para verificar a string recebida, e o código STX para informá-lo de que uma string está sendo enviada.



Figura 16.2 – Tags e cartões RFID podem ter muitos formatos e tamanhos diferentes (imagem cortesia de Timo Arnall).

Projeto 45 – Sistema de controle de acesso

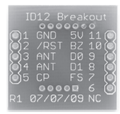
Agora você criará um sistema de controle de acesso. Nele, tags serão lidas utilizando o leitor RFID, e validadas para permitir a abertura de uma porta. O Arduino, por meio de um transistor, operará uma fechadura elétrica.

Componentes necessários

Leitor RFID ID-12



Placa breakout para o ID-12*



Resistor limitador de corrente



LED de 5 mm

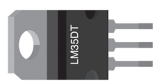


Tags ou cartões de 125 kHz (ao menos 4)

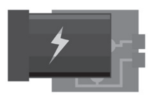
Diodo 1N4001



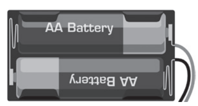
Transistor TIP-120 NPN



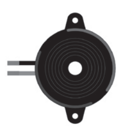
Conector de alimentação de 2,1 mm



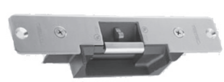
Fonte de alimentação CC de 12 V



Receptor acústico ou alto-falante de 8 Ω



Fechadura elétrica de 12 V



* imagem cortesia da Sparkfun

Conectando os componentes

Conecte tudo como na figura 16.3.

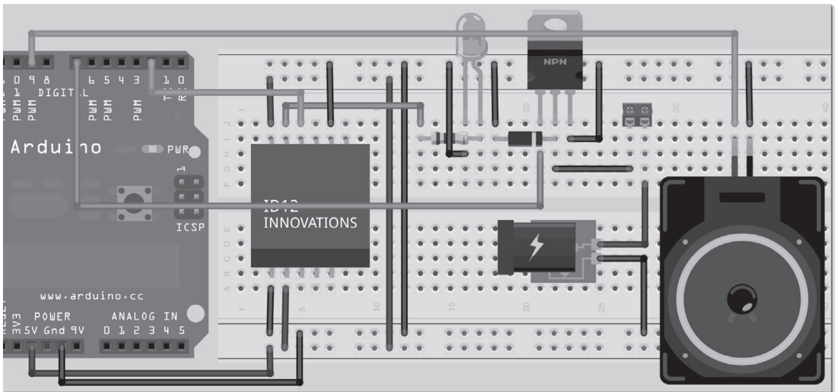


Figura 16.3 – Circuito para o Projeto 45 – Sistema de controle de acesso (consulte o site da Novatec para versão colorida).

Se você não estiver utilizando o TIP120, não deixe de ler o datasheet de seu transistor para garantir a conexão correta da fiação. Da esquerda para a direita no TIP120, você tem a base, o coletor e o emissor. A base vai para o pino digital 7, o coletor vai para o terra (por meio do diodo) e também para o terminal negativo do piezo ou alto-falante. O emissor vai para o terra. Um alto-falante de 8 Ω, se você puder adquiri-lo, produz um som mais alto e agradável do que um piezo.

A alimentação da fechadura deve vir de uma fonte de alimentação CC externa de 12 V, com carga mínima de 500 mA.

Digite o código

Digite o código da listagem 16.2.

Listagem 16.2 – Código para o projeto 45

```
// Projeto 45

#define lockPin 7
#define speakerPin 9
#define tx 3
#define rx 2
#define unlockLength 2000

#include <SoftwareSerial.h>

SoftwareSerial rfidReader = SoftwareSerial(rx, tx);

int users = 3;
```

```
char* cards[] = {  // cartões válidos
  "3D00768B53",
  "3D00251C27",
  "3D0029E6BF",
};

char* names[] = {  // nomes dos portadores dos cartões
  "Tom Smith",
  "Dick Jones",
  "Harry Roberts"
};

void setup() {
  pinMode (lockPin, OUTPUT);
  pinMode (speakerPin, OUTPUT);
  digitalWrite(lockPin, LOW);
  Serial.begin(9600);
  rfidReader.begin(9600);
}

void loop() {
  char cardNum[10];    // array para armazenar o número do cartão
  byte cardBytes[6];   // versão em byte do número do cartão + checksum
  int index=0;         // dígito atual
  byte byteIn=0;       // byte lido do RFID
  byte lastByte=0;     // último byte lido
  byte checksum = 0;   // o resultado do checksum é armazenado aqui

  if (rfidReader.read()==2) {  // lê o leitor RFID
    while(index<12) {  // 12 dígitos em um número serial único
      byteIn = rfidReader.read(); // armazena o valor em byteIn
      if ((byteIn==2) || (byteIn==3) || (byteIn==10) || (byteIn==13))
        {return;}     // se STX, ETX, CR ou LF, interrompe
      if (index<10) {cardNum[index]=byteIn;}   // armazena apenas os primeiros 10 caracteres
                                               // ASCII (os últimos dois indicam o checksum)

      // converte o caractere ASCII para um valor inteiro
      if ((byteIn>='0') && (byteIn<='9')) {
        byteIn -= '0';
      }
      else if ((byteIn>='A') && (byteIn<='F')) {
        byteIn = (byteIn+10)-'A';
      }
      if ((index & 1) == 1) {  // se o número for ímpar, junte dois dígitos de quatro bits em
                               // um byte de 8 bits
        cardBytes[index/2]= (byteIn | (lastByte<<4));   // movimenta o ultimo dígito 4 bits
                                                       // para a esquerda e adiciona um novo dígito

      if (index<10) {checksum ^= cardBytes[index/2];} // calcula o valor total de checksum
    }
```

```
      lastByte=byteIn; // armazena o último byte lido
      index++; // incrementa o índice
      if (index==12) {cardNum[10] = '\0';}  // se atingiu o fim de todos os dígitos, adiciona um
                                            // terminador nulo
    }
    Serial.println(cardNum); // imprime o número do cartão
    int cardIndex =checkCard(cardNum);    // verifica se o cartão é válido e retorna o número do
                                          // índice
    if (cardIndex>=0 && (cardBytes[5]==checksum)) {// se o número do cartão e o checksum forem válidos
      Serial.println("Card Validated");
      Serial.print("User: ");
      Serial.println(names[cardIndex]); // imprime o nome relevante
      unlock();// destrava a porta
      Serial.println();
    }
    else {
      Serial.println("Card INVALID");
      tone(speakerPin, 250, 250);
      delay(250);
      tone(speakerPin, 150, 250);
      Serial.println();
    }
    }
  }
  int checkCard(char cardNum[10]) {
    for (int x=0; x<=users; x++) {    // verifica todos os cartões válidos
      if(strcmp(cardNum, cards[x])==0) {    // compara com o último número de cartão lido
        return (x);   // retorna o índice do número do cartão
      }
    }
    return (-1);    // valores negativos indicam que não houve correspondência
  }
  void unlock() {
    tone(speakerPin, 1000, 500);
    digitalWrite(lockPin, HIGH);
    delay(unlockLength);
    digitalWrite(lockPin, LOW);
  }
```

Certifique-se de que os códigos numéricos de três de suas tags foram digitados no array `cards[]`, no início do programa. Se necessário, utilize o projeto 44 para descobrir esses números (execute o código e abra o monitor serial). Agora, apresente seus quatro cartões ao leitor. Ele piscará o LED para indicar a leitura dos cartões, e você verá uma saída semelhante a esta:

```
3D00251C27
Card Validated
User: Dick Jones

3D0029E6BF
Card Validated
User: Harry Roberts

3D002A7C6C
Card INVALID

3D00768B53
Card Validated
User: Tom Smith
```

O número do cartão será exibido seguido de uma mensagem `"Card INVALID"` (cartão inválido) ou `"Card validated"` (cartão validado), acompanhada do nome de usuário correspondente. Se o cartão for válido, um som agudo será emitido, e a fechadura abrirá por dois segundos. Se o cartão for inválido, você ouvirá um beep duplo e grave, e a porta não será destrancada. A fechadura elétrica de 12 V é alimentada utilizando-se um transistor para oferecer uma tensão e corrente maiores do que as que o Arduino pode fornecer. Você utilizou esse mesmo princípio no capítulo 5, quando controlou um motor CC. Vejamos como este projeto funciona.

Projeto 45 – Sistema de controle de acesso – Análise do código

Primeiramente, temos algumas definições para os pinos que utilizaremos na fechadura e no alto-falante. Da mesma forma, você está utilizando a biblioteca `SoftwareSerial.h`, em vez dos pinos seriais normais nos pinos digitais 0 e 1, por isso deve definir os pinos rx e tx. Você também apresenta um intervalo, em microssegundos, durante o qual a fechadura ficará aberta.

```
#define lockPin 7
#define speakerPin 9
#define tx 3
#define rx 2
#define unlockLength 2000
```

Você utiliza a biblioteca `SoftwareSerial.h` (agora parte do IDE do Arduino) por conveniência. Se estivesse utilizando os pinos rx e tx regulares, teria de desconectar componentes ligados a eles sempre que quisesse fazer o upload de algum código para o Arduino. Empregando a biblioteca `SoftwareSerial.h`, você pode utilizar o pino que quiser.

```
#include <SoftwareSerial.h>
```

Em seguida, você cria uma instância de um objeto `SoftwareSerial`, dando-lhe o nome `rfidReader`. A ela você passa os pinos rx e tx que definiu.

```
SoftwareSerial rfidReader = SoftwareSerial(rx, tx);
```

Depois, temos uma variável para armazenar o número de usuários no banco de dados:

```
int users = 3;
```

Na sequência, temos dois arrays para armazenar os números de identificação dos cartões e os nomes de seus portadores. Altere os números dos cartões para aqueles de sua escolha (apenas os primeiros dez dígitos).

```
char* cards[] = { // cartões válidos
  "3D00768B53",
  "3D00251C27",
  "3D0029E6BF",
};

char* names[] = { // nomes dos portadores dos cartões
  "Tom Smith",
  "Dick Jones",
  "Harry Roberts"
};
```

A rotina de inicialização define os pinos da fechadura e do alto-falante como saída:

```
pinMode (lockPin, OUTPUT);
pinMode (speakerPin, OUTPUT);
```

Depois, definimos o pino da fechadura como `LOW`, para garantir que ela não destranque no início:

```
digitalWrite(lockPin, LOW);
```

Então, você inicia a comunicação na porta serial e na porta `SoftwareSerial`:

```
Serial.begin(9600);
rfidReader.begin(9600);
```

Em seguida, temos o loop principal. Você inicia definindo as variáveis que utilizará no loop:

```
char cardNum[10];    // array para armazenar o número do cartão
byte cardBytes[6];   // versão em byte do número do cartão + checksum
int index=0;         // dígito atual
byte byteIn=0;       // byte lido do RFID
byte lastByte=0;     // último byte lido
byte checksum = 0;   // o resultado do checksum é armazenado aqui
```

Depois, você verifica se há dados entrando pela porta serial dos leitores RFID. Se afirmativo, e o primeiro byte contiver `2` (código de início de transmissão), você saberá que uma string de identificação está prestes a ser transmitida, e poderá iniciar a leitura dos dígitos.

```
if (rfidReader.read()==2) { // lê o leitor RFID
```

Então, temos um loop `while` que será executado enquanto o índice for menor que `12`:

```
While (index<12) {// 12 dígitos em um número serial único
```

A variável `index` armazenará o valor de sua posição atual no dígito que está sendo lido. Como você está lendo um dígito de 12 caracteres de extensão, tem de ler apenas os 12 primeiros dígitos.

Na sequência, o valor da porta serial é lido e armazenado em `byteIn`:

```
byteIn = rfidReader.read();  // armazena o valor em byteIn
```

Apenas no caso de alguns caracteres terem sido esquecidos, você verifica a ocorrência dos códigos de início e término da transmissão, além da presença de caracteres de retorno de carro ou alimentação de linha. Se eles forem detectados, encerra-se o loop.

```
if ((byteIn==2) || (byteIn==3) || (byteIn==10) || (byteIn==13))
  {return;}     // se STX, ETX, CR ou LF, interrompe
```

Os dois últimos caracteres da string de 12 caracteres correspondem ao checksum. Você não deseja armazenar essa informação no array `cardNum`, por isso armazena apenas os primeiros dez caracteres:

```
if (index<10) {cardNum[index]=byteIn;}   // armazena apenas os primeiros 10 caracteres (os
                                        // últimos dois indicam o checksum)
```

A seguir, você tem de converter os caracteres ASCII que está lendo em números hexadecimais equivalentes, por isso executa uma instrução `if-else` para determinar se esses caracteres estão entre `0` e `9`, e `A` e `F`. Se afirmativo, você os converte em seus equivalentes hexadecimais.

```
if ((byteIn>='0') && (byteIn<='9')) {
  byteIn -= '0';
}
else if ((byteIn>='A') && (byteIn<='F')) {
  byteIn = (byteIn+10)-'A';
}
```

Operadores lógicos `E` (`&&`) são utilizados para garantir que os caracteres fiquem entre `0` e `9`, ou entre `A` e `F`. Em seguida, você converte os dois últimos dígitos hexadecimais em um byte. Um dígito hexadecimal tem base dezesseis. O sistema numérico que utilizamos normalmente é base 10, com dígitos de 0 a 9. No sistema hexadecimal, os dígitos vão de 0 a 15. As letras de A a F são utilizadas para representar os números de 10 a 15. Assim, o número FF em hexadecimal corresponde a 255 em decimal:

```
F = 15
(F * 16) + F = (15 * 16) + 15 = 255
```

Dessa forma, você tem de converter os dois últimos dígitos ASCII em um único byte e, depois, em seu equivalente decimal. Você já declarou uma variável `lastByte`, que

armazena o último dígito processado na execução mais recente do loop `while`. Essa variável é inicialmente definida como 0. Isso deve ser feito apenas a cada dois dígitos, uma vez que dois dígitos hexadecimais formam um único byte. Por isso, você confere se o índice é um número ímpar, executando uma operação bit a bit `E` (`&`), que compara o valor armazenado em `index` e o número `1`, e verifica se o resultado também é `1`. Lembre-se de que o índice inicia como 0, por isso o segundo dígito tem índice `1`.

```
if ((index & 1) == 1) {  // se o número for ímpar, junta dois dígitos de quatro bits em um byte
                         // de 8 bits
```

Qualquer número ímpar, quando colocado em uma operação bit a bit E com o número 1, resultará em 1, e qualquer número par, em 0:

```
  12 (par) & 1 = 0
  00001100
  00000001 &
= 00000000

  11 (ímpar) & 1 = 1
  00001011
  00000001 &
= 00000001
```

Caso o resultado determine que você está na posição do segundo, quarto, sexto, oitavo ou décimo segundo dígito, você armazena em `cardBytes` o resultado do seguinte cálculo:

```
cardBytes[index/2]= (byteIn | (lastByte<<4)); // movimenta o ultimo dígito 4 bits para
esquerda, e adiciona um novo dígito
```

Você utiliza `index/2` para determinar o número do índice. Como `index` é um inteiro, apenas o valor antes do ponto decimal será mantido. Assim, para cada dois dígitos que o índice incrementa, o valor do array `cardBytes` se eleva em uma unidade.

O cálculo pega o valor do último byte e o desloca quatro posições para a esquerda. Depois, ele executa nesse número uma operação bit a bit OU (|) com o valor atual lido. Na prática, essa operação pega o primeiro valor hexadecimal, que corresponde aos quatro primeiros bits do número, e desloca esse valor quatro posições para a esquerda. Em seguida, esse número é combinado ao segundo dígito hexadecimal para resultar no byte completo. Assim, se o primeiro dígito fosse `9` e o segundo `E`, o cálculo seria:

```
Lastbyte = 9 =  00001001
00001001 << 4 = 10010000
E = 14 =        00001110
                10010000 OR
=               10011110
```

O checksum é um número que você utiliza para verificar se a string inteira foi lida corretamente. Checksums são muito utilizados em transmissões de dados, e

representam simplesmente o resultado de uma operação XOU sobre cada um dos bytes da string inteira:

```
if (index<10) {checksum ^= cardBytes[index/2];} // calcula o valor total de checksum
```

O número de ID de seu primeiro cartão é:

```
3D00768B53
```

Portanto, o checksum será:

```
3D XOR 00 XOR 76 XOR 8B XOR 53
3D = 00111101
00 = 00000000
76 = 01110110
8B = 10001011
53 = 01010011
```

Se você aplicar uma operação XOU (ou exclusivo) sobre cada um dos dígitos e compará-los, terminará com o valor `93`. Assim, 93 é o checksum para esse número de ID. Se algum dos dígitos tiver sido transmitido incorretamente por interferência, o checksum terá um valor diferente de 93, indicando que o cartão não foi lido corretamente e que você deve desconsiderá-lo.

Fora desse loop, definimos `lastByte` como o valor atual, para que você tenha uma cópia dele na próxima vez.

```
lastByte=byteIn;// armazena o último byte lido
```

Depois, incrementa-se o número de `index`:

```
index++; // incrementa o índice
```

Se você atingiu o fim da string, deve garantir que o décimo dígito no array `cardNum` esteja definido como o código ASCII para `\0`, caractere terminador nulo. Isso é necessário, pois futuramente você terá de verificar se o término da string foi ou não atingido. O terminador nulo indica que você está no fim da string.

```
if (index==12) {cardNum[10] = '\0';}
```

Então, você imprime o número do cartão lido pelo leitor RFID:

```
Serial.println(cardNum); // imprime o número do cartão
```

Depois, um inteiro `cardIndex` é criado e definido com o valor retornado da função `checkCard()` (a ser explicada em breve). Essa função retornará um valor positivo se o número do cartão for válido no banco de dados, e um valor negativo, em caso contrário.

```
int cardIndex =checkCard(cardNum);   // verifica se o cartão é válido e retorna o número do índice
```

Em seguida, você verifica se o número é positivo, e também se o checksum está correto. Se afirmativo, o cartão foi lido corretamente e é válido, por isso você pode destrancar a porta.

```
if(cardIndex>=0 && (cardBytes[5]==checksum)) {// se o número do cartão e o checksum forem válidos
  Serial.println("Card Validated");
  Serial.print("User: ");
  Serial.println(names[cardIndex]);   // imprime o nome relevante
  unlock();  // destrava a porta
  Serial.println();
}
```

Caso o cartão não seja válido, ou caso seu checksum esteja incorreto, ele será considerado inválido e informaremos o usuário do que ocorreu:

```
else {
  Serial.println("Card INVALID");
  tone(speakerPin, 250, 250);
  delay(250);
  tone(speakerPin, 150, 250);
  Serial.println();
}
```

Depois, temos a função `checkCard()`, que retorna um inteiro. Seu parâmetro é o número do cartão passado a ela:

```
int checkCard(char cardNum[10]) {
```

Em seguida, você percorre cada um dos cartões no banco de dados, e verifica se algum deles corresponde ao número de cartão lido:

```
for (int x=0; x<=users; x++) {    // verifica todos os cartões válidos
```

Você utiliza uma função `strcmp`, de comparação de strings, para verificar se o número de cartão informado à função `checkCard()` corresponde ao número na posição atual do banco de dados. É por isso que você necessita de um terminador nulo ao final de seu número, uma vez que a função `srtcmp` o exige.

```
if(strcmp(cardNum, cards[x])==0) {   // compara com o ultimo número de cartão lido
```

A função `strcmp` requer dois parâmetros: as duas strings que você deseja comparar. O número retornado pela função será zero se ambas forem idênticas. Um número diferente de zero indica que elas não correspondem. Caso correspondam, você retorna o valor de `x`, o índice no banco de dados referente ao nome de usuário e ID de cartão válidos.

```
return (x);   // retorna o índice do número do cartão
```

Se os cartões não correspondem, você retorna `-1`.

```
return (-1);    // valores negativos indicam que não houve correspondência
```

A função final é `unlock()`, cujo propósito é tocar um som agudo, destrancar a porta, aguardar um intervalo de tempo predefinido e trancar a porta novamente:

```
void unlock() {
  tone(speakerPin, 1000, 500);
  digitalWrite(lockPin, HIGH);
  delay(unlockLength);
  digitalWrite(lockPin, LOW);
}
```

O próximo passo seria adicionar mais leitores e fechaduras, protegendo sua casa inteira. Usuários autorizados carregariam um cartão ou uma tag que lhes permitisse entrar em quartos relevantes. Direitos individuais de acesso poderiam ser atribuídos a cada usuário, para que tivessem acesso a setores distintos da residência, permitindo apenas usuários válidos em áreas separadas.

Agora, vamos para o capítulo final do livro, no qual conectaremos seu Arduino à Internet.

Resumo

Neste capítulo, você viu como é fácil ler dados de um cartão ou uma tag RFID, e como utilizar esses dados para destrancar uma fechadura elétrica ou realizar algo diferente. Já encontrei projetos em que se prende um pingente RFID a várias chaves e coloca-se um leitor RFID em um recipiente. Posteriormente, quando o usuário volta para casa, ele arremessa suas chaves dentro desse recipiente. A casa responde à identificação da chegada da pessoa, regulando seus níveis preferidos de luminosidade e temperatura, tocando sua música favorita, ligando o chuveiro etc. Quando falamos em um leitor RFID, o único limite é sua imaginação.

Assuntos e conceitos abordados no capítulo 16:

- como funciona a tecnologia RFID;
- como conectar um leitor RFID ID-12 a um Arduino;
- como ler dados seriais a partir de um leitor RFID;
- como utilizar um transistor para controlar um dispositivo de maior potência;
- como utilizar a biblioteca `SoftwareSerial`;
- como converter valores ASCII em hexadecimal;
- como utilizar o E bit a bit para verificar se um número é ímpar ou par;
- como mesclar dois dígitos hexadecimais, utilizando deslocamentos de bits e operações OU bit a bit, para criar um byte;
- como criar checksums utilizando operações XOU (OU exclusivo);
- como utilizar a função `strcmp` para comparar duas strings.

CAPÍTULO 17

Comunicação via Ethernet

Para este capítulo final, você verá como conectar seu Arduino a um roteador e transmitir dados por um cabo Ethernet. Ao fazê-lo, você poderá ler as informações do Arduino a partir de outro ponto em sua rede. Você também poderá enviar dados para a Internet, tornando o conteúdo visível em um navegador. Nos projetos que veremos, você utilizará o shield Ethernet oficial do Arduino.

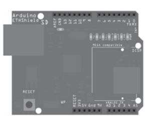
A capacidade de conectar seu Arduino a uma rede ou à Internet torna possível muitos projetos diferentes. Você pode enviar dados para sites, como atualizações no Twitter. Também poderá controlar o Arduino pela Internet, ou utilizá-lo como um servidor web para disponibilizar páginas simples contendo dados de sensores, e assim por diante. Este capítulo fornecerá o conhecimento básico para que você possa criar seus próprios projetos do Arduino com base na Internet ou Ethernet.

Projeto 46 – Shield Ethernet

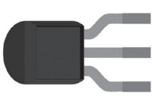
Agora você utilizará o shield Ethernet e alguns sensores de temperatura para demonstrar como acessar o Arduino pela Ethernet.

Componentes necessários

Shield Ethernet do Arduino



2 sensores de temperatura DS18B20



Resistor de 4,7 kΩ



Conectando os componentes

Insira o shield Ethernet sobre o Arduino e conecte tudo como na figura 17.1. Os fios devem ser ligados ao shield nos mesmos locais em que seriam em uma placa Arduino.

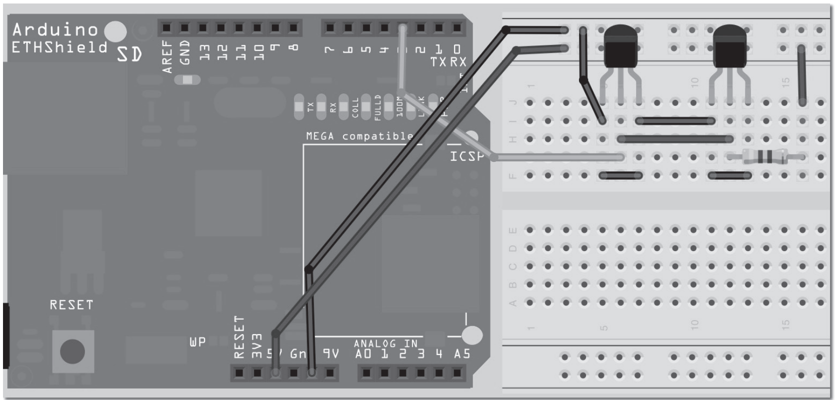


Figura 17.1 – Circuito para o Projeto 46 – Shield Ethernet (consulte o site da Novatec para versão colorida).

Digite o código

Digite o código da listagem 17.1.

Listagem 17.1 – Código para o projeto 46

```
// Projeto 46 - Baseado no exemplo Arduino Webserver, criado por David A. Mellis e Tom Igoe

#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Fio de dados é plugado ao pino 3 do Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float tempC, tempF;

// Prepara uma instância de OneWire para se comunicar com qualquer dispositivo 1-Wire (não
// apenas com CIs de temperatura Maxim/Dallas)
OneWire oneWire(ONE_WIRE_BUS);
// Passa o endereço da instância oneWire ao Dallas Temperature.
DallasTemperature sensors(&oneWire);

// arrays para armazenar os endereços dos dispositivos
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};

byte mac[] = { 0x48, 0xC2, 0xA1, 0xF3, 0x8D, 0xB7 };
byte ip[] = { 192, 168 ,0, 104 };

// Inicia o servidor na porta 80
Server server(80);
```

```
void setup() {
  // Inicia a ethernet e o servidor
  Ethernet.begin(mac, ip);
  server.begin();
  // Inicaliza a biblioteca sensors
  sensors.begin();
  // define a resolução
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
}

// função para obter a temperatura de um dispositivo
void getTemperature(DeviceAddress deviceAddress) {
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}
void loop() {
  sensors.requestTemperatures();

  // escuta a entrada de clientes
  Client client = server.available();
  if (client) {
    // uma solicitação http termina com uma linha em branco
    boolean BlankLine = true;
    while (client.connected()) {
      if (client.available()) {
      char c = client.read();

      // Se a linha estiver em branco e o fim da linha for um caractere newline '\n' = fim da
      // solicitação HTTP
      if (c == '\n' && BlankLine) {
        getTemperature(insideThermometer);
        client.println("HTTP/1.1 200 OK");    // Resposta HTTP padrão
        client.println("Content-Type: text/html\n");
        client.println("<html><head><META HTTP-EQUIV=""refresh""CONTENT=""5"">\n");
        client.println("<title>Arduino Web Server</title></head>");
        client.println("<body>\n");
        client.println("<h1>Arduino Web Server</h1>");
        client.println("<h3>Internal Temperature</h3>");
        client.println("Temp C:");
        client.println(tempC);
        client.println("<br/>");
        client.println("Temp F:");
        client.println(tempF);
        client.println("<br/>");
        getTemperature(outsideThermometer);
        client.println("<h3>External Temperature</h3>");
        client.println("Temp C:");
```

```
          client.println(tempC);
          client.println("<br/>");
          client.println("Temp F:");
          client.println(tempF);
          client.println("<br/>");

          break;
      }
        if (c == '\n') {
          // Iniciando uma nova linha
          BlankLine = true;
      }
        else if (c != '\r') {
          // A linha atual tem um caractere nela
          BlankLine = false;
        }
      }
    }
    // Permite um intervalo de tempo suficiente para que o navegador receba os dados
    delay(10);
    // Fecha a conexão
    client.stop();
    }
  }
```

Você terá de digitar os dois números de endereço dos sensores de temperatura (consulte o projeto 37) nesta linha:

```
byte ip[] = { 192, 168, 0, 104 };
```

Você também terá de alterar o endereço IP digitando o seu próprio endereço. Para fazê-lo, é necessário que você descubra, a partir de seu roteador, qual endereço de IP foi reservado para os dispositivos em seu computador. Geralmente, o endereço iniciará com 192.168.0 ou 192.168.1 — basta adicionar mais um número, tipicamente maior que 100, para garantir que ele não interfira nos dispositivos existentes. Talvez você também tenha de acessar as configurações de seu roteador, para garantir que quaisquer solicitações HTTP feitas à porta 80 sejam encaminhadas para o endereço IP do shield Ethernet. Consulte a seção "Port Forwarding" (encaminhamento de porta) no manual de seu roteador. Também pode ser necessário abrir a porta 80 em seu firewall.

Agora, abra seu navegador web e digite o endereço IP e a porta, por exemplo:

```
192.168.0.104:80
```

Se tudo estiver funcionando corretamente, você verá a página web da figura 17.2 em seu navegador.

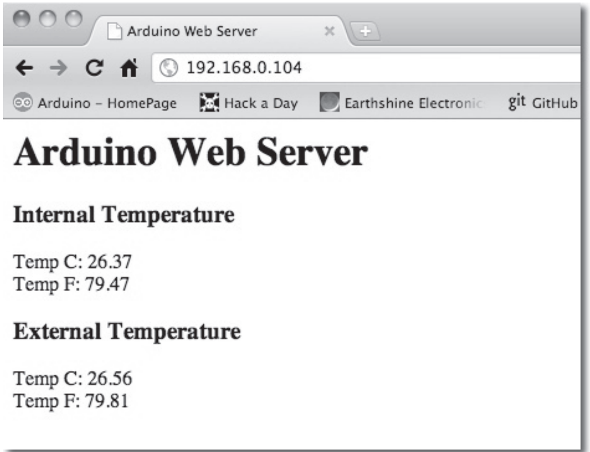


Figura 17.2 – Saída do navegador web, produzida pelo servidor web do Arduino.

A página se atualizará automaticamente a cada cinco segundos para mostrar alterações nas temperaturas. Caso você tenha configurado corretamente o encaminhamento de porta e o firewall em seu roteador, você também poderá acessar essa página a partir de qualquer ponto com acesso à Internet. Para isso, você deverá saber o endereço IP do roteador, o qual pode ser encontrado na página de administração do roteador. Digite-o, seguido pelo número da porta, em qualquer navegador, como o exemplo que vemos a seguir:

```
95.121.118.204:80
```

A página que mostramos será exibida no navegador e você poderá verificar as leituras de temperatura a partir de qualquer local com acesso à Internet.

Projeto 46 – Shield Ethernet – Análise do código

Algumas partes desse código são repetidas do projeto 37, por isso veremos rapidamente essas seções, dando preferência e nos concentrando nas partes relacionadas ao shield Ethernet. Primeiramente, você carrega as bibliotecas. Certifique-se de que você colocou as bibliotecas para os sensores de temperatura em sua pasta `libraries` (Projeto 37). Note que, a partir da versão 0019 do Arduino IDE, é necessário incluir a biblioteca `SPI.h` em qualquer projeto que utilize a biblioteca `Ethernet.h`.

```
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Em seguida, definimos o pino e a precisão dos sensores:

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

Também definimos duas variáveis `float` para armazenar a temperatura em Celsius e Fahrenheit:

```
float tempC, tempF;
```

Uma instância de `OneWire` (chamada `oneWire`) é criada, e você passa uma referência dela à biblioteca Dallas Temperature:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

Os endereços para os dois sensores de temperatura são definidos. Lembre-se de verificá-los utilizando o código do projeto 37, se necessário.

```
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};
```

Em seguida, você tem de definir os endereços MAC e IP do dispositivo:

```
byte mac[] = { 0x48, 0xC2, 0xA1, 0xF3, 0x8D, 0xB7 };
byte ip[] = { 192,168,0, 104 };
```

O endereço MAC (Media Access Control) é um identificador único para interfaces de rede. A placa de rede em seu PC ou Mac terá seu endereço MAC definido pelo fabricante. No seu caso, você mesmo está decidindo qual deve ser esse endereço. Trata-se simplesmente de um número de 48 bits, por isso utilize quaisquer seis dígitos hexadecimais no campo do endereço, se bem que não há problema em deixá-lo como está. O endereço IP terá de ser manualmente definido, e deve estar dentro do intervalo permitido pelo seu roteador.

Em seguida, uma instância de um objeto de servidor é criada, acompanhada do número da porta do dispositivo:

```
Server server(80);
```

O servidor escutará conexões de entrada na porta especificada. Um número de porta é simplesmente uma via para os dados. Você tem apenas um cabo Ethernet conectado ao seu dispositivo, mas o número de porta decide para onde os dados devem ir. Imagine o endereço MAC como o endereço de um grande prédio de apartamentos, e o número de porta como o número individual de cada apartamento.

Depois, temos a rotina de inicialização. Você começa inicializando a comunicação Ethernet e passando os endereços MAC e IP do dispositivo para a instância:

```
Ethernet.begin(mac, ip);
```

Agora, você tem de dizer ao seu servidor que comece a escutar a entrada de conexões utilizando o comando `begin()`:

```
server.begin();
```

Você também deve iniciar seus sensores e definir suas resoluções:

```
sensors.begin();
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

Em seguida, você cria a função responsável por obter as temperaturas do sensor (como no projeto 37):

```
void getTemperature(DeviceAddress deviceAddress) {
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}
```

Depois, temos o loop principal do programa. Primeiramente, você solicita as temperaturas dos dois sensores:

```
sensors.requestTemperatures();
```

Você tem de escutar a entrada de clientes, como páginas web que solicitam a visualização da página servida pelo Arduino. Para tanto, você cria uma instância do tipo `Client` e verifica se há dados disponíveis para leitura a partir do servidor. O cliente é o navegador que se conectará ao Arduino. O servidor é o Arduino.

```
Client client = server.available();
```

Em seguida, você verifica se um cliente se conectou, e se há dados disponíveis para ele. Se verdadeiro, a instrução `if` é executada.

```
if (client) {
```

Primeiro, a instrução `if` cria uma variável booleana, `BlankLine`, e a define como `true`:

```
boolean BlankLine = true;
```

Uma solicitação HTTP do cliente terminará com uma linha em branco, tendo ao seu final um caractere de nova linha. Assim, você utiliza a variável `BlankLine` para determinar se atingiu ou não o fim dos dados.

Em seguida, você verifica se o cliente ainda está conectado e, se afirmativo, executa o código dentro do loop `while`:

```
while (client.connected()) {
```

Depois, você verifica se os dados estão ou não disponíveis ao cliente. Se os dados estiverem disponíveis, o código da próxima instrução `if` será executado. O comando `available()` retorna o número de bytes que o servidor escreveu no cliente. Se o valor for maior que zero, a instrução `if` será executada.

```
if (client.available()) {
```

Então, uma variável de tipo `char` é criada para armazenar o próximo byte recebido do servidor. Utilize o comando `client.read()` para obter o byte.

```
char c = client.read();
```

Se o caractere lido for um caractere de nova linha ("\n"), você também deverá verificar se `BlankLine` é ou não verdadeiro. Se afirmativo, isso indicará que você atingiu o fim da solicitação HTTP e pode, portanto, servir o código HTML para o cliente (o navegador do usuário).

```
if (c == '\n' && BlankLine) {
```

Em seguida, temos os dados que serão enviados a partir de seu servidor. Você inicia obtendo a temperatura de seu sensor interno.

```
getTemperature(insideThermometer);
```

Depois, temos o código HTML que deve ser emitido para o cliente. Toda página é formada de código HTML (ou HyperText Markup Language, Linguagem de Marcação de Hipertexto). Explicar essa linguagem está além do escopo deste livro, por isso fornecerei apenas algumas informações básicas. Se você deseja estudar mais sobre esse tópico, consulte sua entrada na Wikipédia, em *http://en.wikipedia.org/wiki/HTML*. Também podem ser encontrados muitos tutoriais referentes à linguagem HTML na Internet. Você utiliza o comando `client.println()` para emitir dados ao cliente. Basicamente, você envia o código para criar uma página web. Na maioria dos navegadores, caso você clique com o botão direito em uma página web, terá a opção de visualizar o código-fonte. Experimente essa funcionalidade e você verá o código HTML que compõe a página web visualizada. O código diz ao navegador o que ele deve mostrar e como isso deve ser feito.

Primeiramente, você diz ao cliente que está utilizando HTTP versão 1.1, protocolo padrão para emissão de páginas web, e que o conteúdo que você está prestes a enviar é HTML:

```
client.println("HTTP/1.1 200 OK"); // Resposta HTTP padrão
client.println("Content-Type: text/html\n");
```

Em seguida, você utiliza a tag `html` para dizer que tudo, desse ponto em diante, será código HTML; e também a tag `head`, contendo os comandos que você deseja emitir para o navegador, além de scripts que deseja executar etc., antes do corpo principal do código. O primeiro comando diz ao navegador que você deseja que a página recarregue automaticamente a cada cinco segundos.

```
client.println("<html><head><META HTTP-EQUIV=""refresh""CONTENT=""5"">\n");
```

Então, você fornece um título à página, o qual será mostrado no topo do navegador e em suas guias.

```
client.println("<title>Arduino Web Server</title></head>\n");
```

Você finaliza a seção de cabeçalho inserindo uma tag `</head>`. Em seguida, temos o corpo do HTML. Essa é a parte visível ao usuário.

```
client.println("<body>\n");
```

Você exibe um cabeçalho `<h1>`, com os dizeres "Arduino Web Server". O H1 é o maior cabeçalho, seguido de H2, H3 etc.

```
client.println("<h1>Arduino Web Server</h1>");
```

Em seguida, temos o título da próxima seção, "Internal Temperature", como um cabeçalho H3:

```
client.println("<h3>Internal Temperature</h3>");
```

Então, você imprime a temperatura em Celsius e Fahrenheit, seguidas por quebras de linhas `<br/>`.

```
client.println("Temp C:");
client.println(tempC);
client.println("<br/>");
client.println("Temp F:");
client.println(tempF);
client.println("<br/>");
```

Depois, as temperaturas externas são solicitadas e exibidas:

```
getTemperature(outsideThermometer);
client.println("<h3>External Temperature</h3>");
client.println("Temp C:");
client.println(tempC);
client.println("<br/>");
client.println("Temp F:");
client.println(tempF);
client.println("<br/>");
```

Então, encerramos o loop `while` com um comando `break`:

```
break;
```

Agora, você define `BlankLine` como `true`, caso um caractere `\n` (nova linha) seja lido, e `false`, se ele não for um `\r` (retorno de carro), ou seja, se ainda houver caracteres a serem lidos a partir do servidor.

```
if (c == '\n') {
  // Iniciando uma nova linha
  BlankLine = true;
}
else if (c != '\r') {
  // A linha atual tem um caractere nela
  BlankLine = false;
}
```

Então, você espera um pequeno intervalo de tempo, permitindo ao navegador que receba os dados, e interrompe o cliente com um comando `stop()`. Isso desconecta o cliente do servidor.

```
delay(10);
client.stop();
```

Este projeto foi uma introdução básica sobre como servir uma página web, que apresenta dados de sensores incorporados, por meio do shield Ethernet. Certamente há formas mais adequadas de apresentação de dados pela Intenet; você verá uma delas a seguir.

Projeto 47 – Mostrador meteorológico conectado à Internet

Agora você utilizará os mesmos componentes e o mesmo circuito do projeto anterior, mas, dessa vez, enviará os dados de temperatura dos sensores para o Pachube, serviço de banco de dados online que permite aos usuários conectar dados de sensores à web (Figura 17.3.) Dados podem ser enviados em muitos formatos e exibidos em um feed no site, na forma de um gráfico. Os gráficos são construídos em tempo real e podem ser incorporados facilmente aos seus sites. Você também pode visualizar dados históricos, acessados a partir de um feed, assim como enviar alertas para controlar scripts, dispositivos etc. Há uma série de tutoriais no site dedicados ao Arduino.

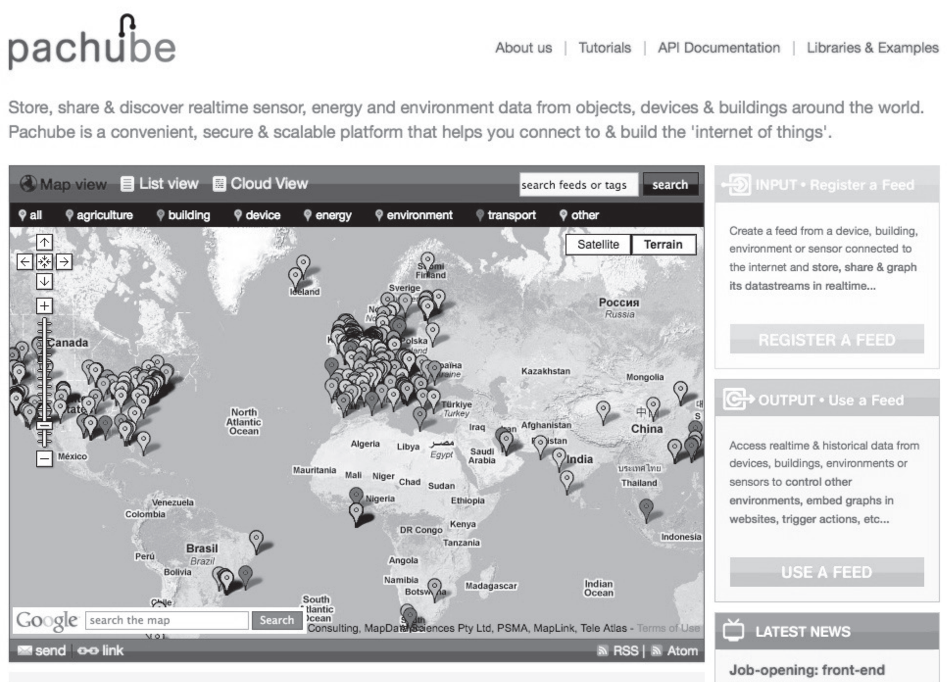


Figura 17.3 – Site do Pachube.

Para utilizar o serviço, você deve registrar uma conta no Pachube. Esse serviço representa a melhor forma de exibir dados de sensores, monitores de energia doméstica, sistemas de monitoramento de imóveis e assim por diante. Há muitos níveis de acesso — desde um serviço gratuito (com dez fluxos de dados (datastreams), dez atualizações por minuto e histórico armazenado de 30 dias), até quatro níveis de serviço pago, os quais permitem mais fluxos de dados, taxas de atualização mais frequentes e históricos mais extensos.

Uma conta deve ser criada antes que você possa fazer o upload de dados, por isso acesse o site *www.pachube.com* e clique no botão “SIGN UP”. Escolha um plano (você pode experimentar gratuitamente qualquer um dos serviços por sete dias). Para este projeto, escolha o serviço gratuito, a primeira opção. Se você gostar dos recursos fornecidos e necessitar de mais fluxos de dados, poderá fazer, posteriormente, um upgrade para um dos serviços pagos.

A página seguinte pede que você forneça um nome de usuário, um endereço de e-mail e uma senha. Preencha esses dados e pressione o botão “SIGN UP”. Completado o registro, será feito seu login e você será conduzido de volta à página principal. Agora você deve criar um feed. Clique no botão “Register a Feed” (registrar um feed). Será necessário preencher um formulário (Figura 17.4).



Figura 17.4 – Página de registro de feed no Pachube.

Escolha um tipo de feed manual e digite seu título. Você também pode escolher uma localização para o feed, assim como digitar dados em sua caixa de descrição, se preferir. Todos os feeds no serviço gratuito do Pachube são públicos, por isso não digite informações que não deseja tornar públicas nessa página.

Depois, adicione quatro feeds (os feeds de dados de seus sensores de temperatura), que exibirão as temperaturas internas e externas em Celsius e Fahrenheit. Deixe os campos de ID como estão e digite os nomes dos feeds, bem como as unidades e símbolos utilizados (Figura 17.4).

Assim que tiver digitado todos os dados, clique no botão “Save Feed” (salvar feed), e você será conduzido de volta à página do feed que acabou de criar. Nela, você verá os nomes dos fluxos de dados digitados na página de registro.

Agora, é necessário obter algumas informações para o seu código. Primeiro, você necessita do número do feed. Veja o canto superior esquerdo da página de seu feed, abaixo do título, e observe o URL que termina em .XML; o número ao final desse URL é a identificação de seu feed (Figura 17.5). Anote esse número.

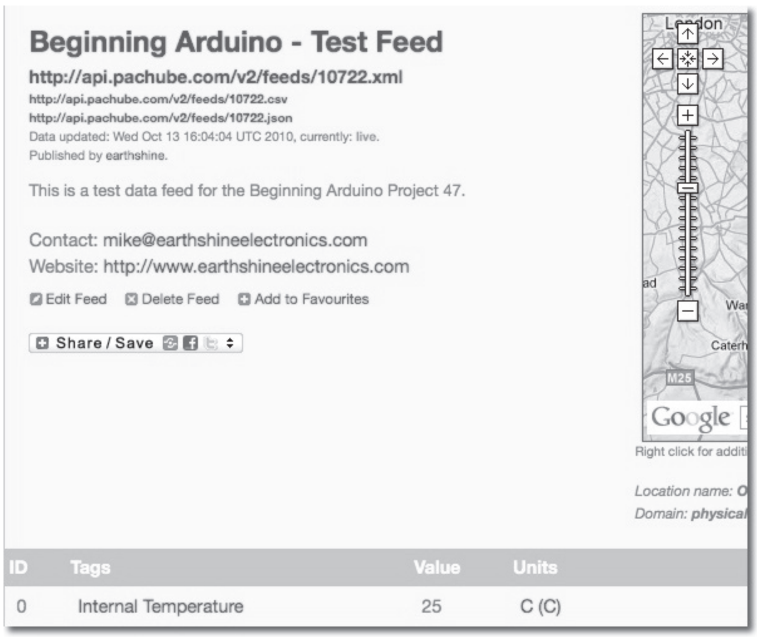


Figura 17.5 – Número do feed abaixo do título.

Em seguida, você necessitará de sua Master API Key (chave mestra de API). Para obter esse dado, clique em “My Settings” (minhas configurações) no topo da página. Um longo número, com o título “Your Master API Key” (sua chave mestra de API) será exibido (Figura 17.6). Copie e cole essa informação no código.

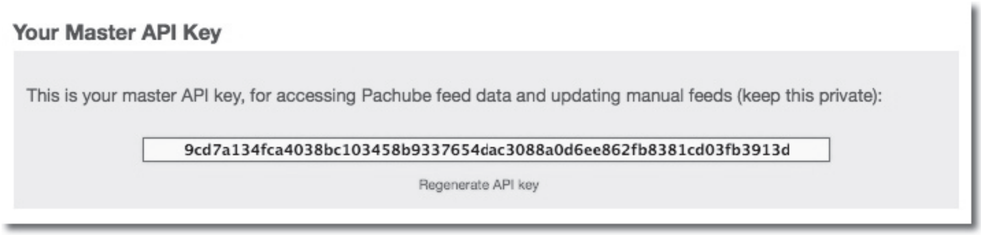


Figura 17.6 – Master API Key.

Tendo essas informações vitais, você pode, agora, digitar o código.

Digite o código

Digite o código da listagem 17.2. Agradecemos a Usman Hague, do Pachube, pelo auxílio neste projeto.

Listagem 17.2 – Código para o projeto 47

```
// Projeto 47 - Baseado nos exemplos do Arduino com o Pachube
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define SHARE_FEED_ID  10722     // Este é o ID de seu feed no Pachube
#define UPDATE_INTERVAL  10000  // Se a conexão for boa, espere 10 segundos antes de atualizar -
                                // não deve ser menor que 5
#define RESET_INTERVAL  10000   // Se a conexão falha/reinicia, espere 10 segundos antes de
                                // tentar novamente - não deve ser menor que 5
#define PACHUBE_API_KEY  "066ed6ea1d1073600e5b44b35e8a399697d66532c3e736c77dc11123dfbfe12f"
// Preencha sua chave para a API

// Fio de dados é plugado ao pino 3 do Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

// Prepara uma instância de OneWire (chamada oneWire) para se comunicar com qualquer
dispositivo 1-Wire (não apenas com CIs de temperatura Maxim/Dallas)
OneWire oneWire(ONE_WIRE_BUS);

// Passa o endereço da instância oneWire ao Dallas Temperature.
DallasTemperature sensors(&oneWire);

// arrays para armazenar os endereços dos dispositivos
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};
byte mac[] = { 0xCC, 0xAC, 0xBE, 0xEF, 0xFE, 0x91 };  // certifique-se de que esse valor é único
                                                      // em sua rede
```

```
byte ip[] = { 192, 168, 0, 104   };                // sem DHCP, por isso definimos nosso
                                                   // próprio endereço IP
byte remoteServer[] = { 173, 203, 98, 29 };        // pachube.com
Client localClient(remoteServer, 80);

unsigned int interval;
char buff[64];
int pointer = 0;
char pachube_data[70];
char *found;
boolean ready_to_update = true;
boolean reading_pachube = false;
boolean request_pause = false;
boolean found_content = false;
unsigned long last_connect;
int content_length;
int itempC, itempF, etempC, etempF;
void setupEthernet() {
  resetEthernetShield();
  delay(500);
  interval = UPDATE_INTERVAL;
  Serial.println("setup complete");
}
void clean_buffer() {
  pointer = 0;
  memset(buff,0,sizeof(buff));
}
void resetEthernetShield(){
  Serial.println("reset ethernet");
  Ethernet.begin(mac, ip);
}
void pachube_out() {
  getTemperatures();
  if (millis() < last_connect) last_connect = millis();
  if (request_pause) {
    if ((millis() - last_connect) > interval){
      ready_to_update = true;
      reading_pachube = false;
      request_pause = false;
    }
  }
  if (ready_to_update) {
    Serial.println("Connecting...");
```

```
    if (localClient.connect()) {
    sprintf(pachube_data,"%d,%d,%d,%d",itempC, itempF, etempC, etempF);
    Serial.print("Sending: ");
    Serial.println(pachube_data);
    content_length = strlen(pachube_data);

    Serial.println("Updating.");

      localClient.print("PUT /v1/feeds/");
      localClient.print(SHARE_FEED_ID);
      localClient.print(".csv HTTP/1.1\nHost: api.pachube.com\nX-PachubeApiKey: ");
      localClient.print(PACHUBE_API_KEY);
      localClient.print("\nUser-Agent: Beginning Arduino – Project 47");
      localClient.print("\nContent-Type: text/csv\nContent-Length: ");
      localClient.print(content_length);
      localClient.print("\nConnection: close\n\n");
      localClient.print(pachube_data);
      localClient.print("\n");

      ready_to_update = false;
      reading_pachube = true;
      request_pause = false;
      interval = UPDATE_INTERVAL;
    }
    else {
      Serial.print("connection failed!");
      ready_to_update = false;
      reading_pachube = false;
      request_pause = true;
      last_connect = millis();
      interval = RESET_INTERVAL;
      setupEthernet();
    }
  }

  while (reading_pachube) {
    while (localClient.available()) {
      checkForResponse();
    }
    if (!localClient.connected()) {
      disconnect_pachube();
    }
  }
}

void disconnect_pachube() {
  Serial.println("disconnecting.\n===============\n\n");
  localClient.stop();
  ready_to_update = false;
```

```
    reading_pachube = false;
    request_pause = true;
    last_connect = millis();
    resetEthernetShield();
  }
  void checkForResponse() {
    char c = localClient.read();
    buff[pointer] = c;
    if (pointer < 64) pointer++;
    if (c == '\n') {
      found = strstr(buff, "200 OK");
      buff[pointer]=0;
      clean_buffer();
    }
  }
  // Função para obter a temperatura de um dispositivo
  void getTemperatures() {
    sensors.requestTemperatures();
    itempC = sensors.getTempC(insideThermometer);
    itempF = DallasTemperature::toFahrenheit(itempC);
    etempC = sensors.getTempC(outsideThermometer);
    etempF = DallasTemperature::toFahrenheit(etempC);
  }
  void setup() {
    Serial.begin(57600);
    setupEthernet();
    // Inicie a biblioteca sensors
    sensors.begin();
    // defina a resolução
    sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
    sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
  }
  void loop() {
    pachube_out();
  }
```

Faça o upload do código e abra o monitor serial. Se tudo estiver funcionando corretamente e você estiver enviando dados ao Pachube, a saída será semelhante à que vemos a seguir:

```
reset ethernet
setup complete
Connecting...
Sending: 25,77,25,77
Updating.
disconnecting.
===============
```

Agora, abra seu navegador web e acesse *www.pachube.com*. Navegue até seu feed e visualize a página. A data e hora em que o feed foi atualizado pela última vez devem aparecer abaixo do título, acompanhadas de um botão “live”, mostrando que o feed está ativo e recebendo dados (Figura 17.7). Os gráficos devem estar exibindo a temperatura ao longo do tempo. Se você deixar o sensor funcionando por tempo suficiente, poderá ver as alterações de temperatura durante o dia.

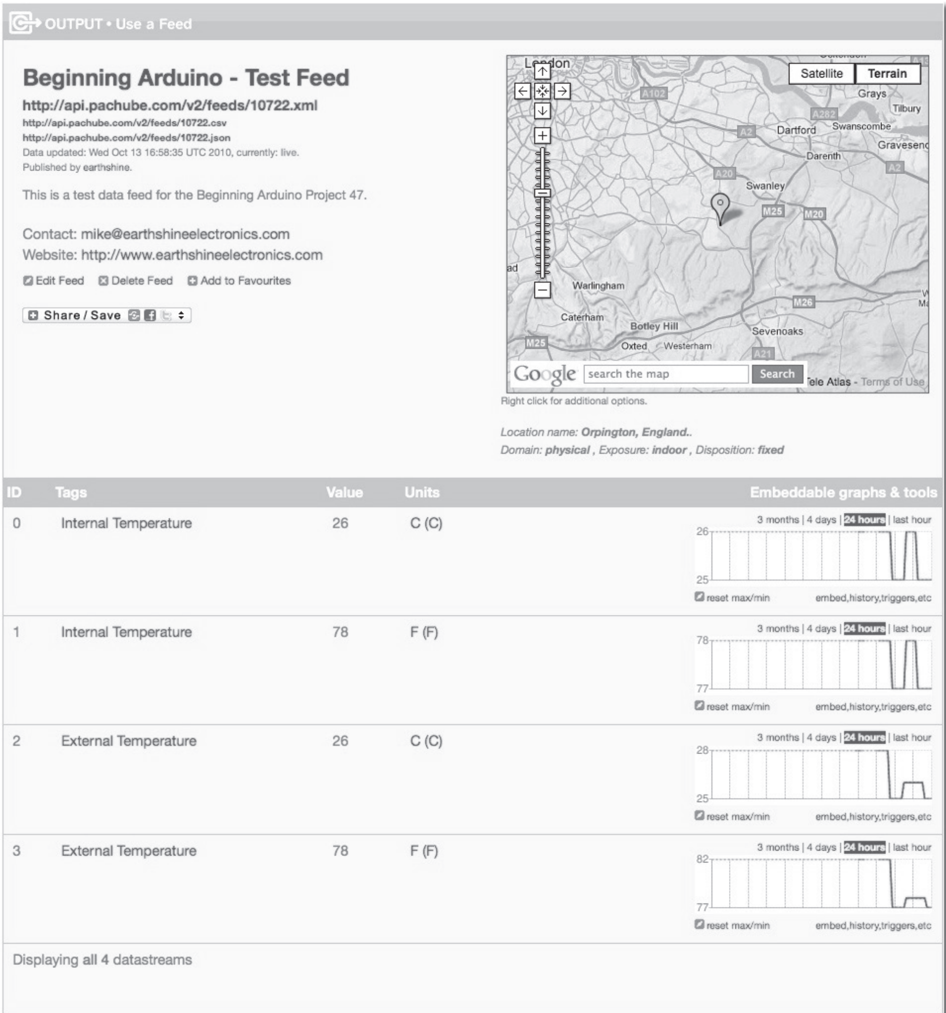


Figura 17.7 – Página de feed ativo no Pachube.

Clicando nos botões acima do gráfico você tem acesso aos dados referentes à última hora, às últimas 24 horas, aos últimos quatro dias e aos últimos três meses. Se você clicar no gráfico, terá acesso a uma nova página, em que os dados são exibidos com resolução máxima. Essas imagens podem ser incorporadas à sua própria página web, fornecendo um feed ativo de dados. Há outros tipos de gráfico no repositório de

aplicativos do Pachube, assim como aplicativos que permitem a visualização de dados utilizando um dispositivo móvel. Os gráficos também podem ser personalizados em termos de tamanho, cor, tipo de grade etc.

Você pode modificar o código para adicionar outros sensores de temperatura, assim como o sensor de pressão que utilizamos no projeto 31. Também pode adicionar sensores que meçam a luminosidade do ambiente, sua umidade, e sensores que medem a velocidade do vento, criando uma estação meteorológica completa, que poderá ser acessada pela Internet no Pachube.

Agora, vejamos o código deste projeto para compreender como tudo funciona.

Projeto 47 – Estação meteorológica conectada à Internet – Análise do código

O código inicia com os `#includes` para o shield Ethernet e os sensores de temperatura OneWire:

```
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Em seguida, o número do feed é definido. Você terá de alterar esse número para que ele corresponda ao ID de seu feed.

```
#define SHARE_FEED_ID  10722
```

Depois, você define, em milissegundos, o intervalo de tempo entre as atualizações de dados no Pachube, assim como o tempo que se deve esperar entre falhas na conexão. Certifique-se de que esses números não estejam abaixo de cinco segundos, ou você receberá um "API key rate-limiting warning" (alerta de limitação de taxa da chave da API). Se você necessita de mais velocidade em suas atualizações, adquira uma das assinaturas pagas do serviço.

```
#define UPDATE_INTERVAL  10000
#define RESET_INTERVAL  10000
```

Agora, você tem de definir sua chave API: aquela da página "my settings". Copie e cole a sequência da página no código:

```
#define PACHUBE_API_KEY  "066ed6eb e5b449c77dc1d13d66532c3e736073605e8a3a 0969711123dfbfe12f"
```

Então, temos os dois `#defines` para o pino e para a precisão dos sensores de temperatura:

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

Seguidos pela criação de uma instância de um barramento 1-Wire e de um objeto `DallasTemperature`:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

Então, definimos os números seriais dos dois sensores DS17B20:

```
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};
```

Depois, você deve digitar os endereços MAC e IP de seu shield Ethernet:

```
byte mac[] = { 0xCC, 0xAC, 0xBE, 0xEF, 0xFE, 0x91 };  // certifique-se de que esse valor seja
                                                        // único em sua rede
byte ip[] = { 192, 168, 0, 104   };          // sem DHCP, por isso definimos nosso próprio
                                             // endereço IP
```

Então, definimos o endereço IP do servidor remoto (endereço IP de Pachube.com):

```
byte remoteServer[] = { 173, 203, 98, 29 };
```

Em seguida, você cria uma instância de `Client`, passando a ela o endereço do Pachube e a porta 80:

```
Client localClient(remoteServer, 80);
```

Depois, temos todas as variáveis que utilizaremos no programa, iniciando com o intervalo em milissegundos que deve haver entre as atualizações ou tentativas de conexão:

```
unsigned int interval;
```

Depois, temos um array para armazenar os caracteres lidos do Pachube:

```
char buff[64];
```

Um ponteiro ou índice para o array anterior:

```
int pointer = 0;
```

Um array para armazenar a string que será enviada ao Pachube (utilize um número maior, caso você esteja enviando strings de dados mais longas):

```
char pachube_data[70];
```

Uma variável para armazenar o resultado de uma função de comparação de strings que veremos mais adiante (utilizada para verificar se os dados foram recebidos corretamente pelo feed):

```
char *found;
```

Uma série de variáveis booleanas autoexplicativas:

```
boolean ready_to_update = true;
boolean reading_pachube = false;
boolean request_pause = false;
boolean found_content = false;
```

O tempo, em milissegundos, desde a última conexão, que será comparado ao intervalo definido e ao valor em `millis()`, para decidir se você deve ou não fazer outra atualização:

```
unsigned long last_connect;
```

O comprimento da string de dados que você terá de enviar ao Pachube:

```
int content_length;
```

E, finalmente, as temperaturas em Celsius e Fahrenheit de seus sensores:

```
int itempC, itempF, etempC, etempF;
```

Em seguida, temos a primeira de uma série de funções utilizadas por todo o sketch. A primeira que encontramos é `setupEthernet()`, a qual reinicializa a conexão Ethernet e atualiza a definição de intervalo:

```
void setupEthernet() {
  resetEthernetShield();
  delay(500);
  interval = UPDATE_INTERVAL;
  Serial.println("setup complete");
}
```

Depois, temos a função `clean_buffer()`, utilizada para limpar o buffer e preenchê-lo com valor 0:

```
void clean_buffer() {
```

Primeiro, o ponteiro (índice do array) é definido como 0:

```
pointer = 0;
```

Depois, você utiliza um comando `memset` para preencher o espaço na memória do buffer com zeros:

```
memset(buff,0,sizeof(buff));
```

O comando `memset` é algo novo para você. Sua função é definir certo número de bytes na memória com um valor específico. Ele requer três parâmetros: um ponteiro para o bloco da memória que deve ser preenchido, o valor a ser usado no preenchimento e o número de bytes que será utilizado.

Em seu caso, você passa `buff` como primeiro parâmetro, assim, ele aponta para o primeiro byte na memória onde `buff` está armazenado. Depois, ele escreve o valor 0 nesse bloco de memória e em todos os blocos até `sizeof(buff)`. O comando `sizeof()` retorna o tamanho do array em bytes.

O propósito de `memset` foi preencher com 0 a memória ocupada pelo array `buff`, limpando-a de todos os dados escritos previamente.

Em seguida, temos a função `resetEthernetShield()`, cuja função é reinicializar o shield Ethernet, executando um comando `Ethernet.begin` para reinicializar o shield cada vez que ela é chamada:

```
void resetEthernetShield() {
  Serial.println("reset ethernet");
  Ethernet.begin(mac, ip);
}
```

Depois, temos uma extensa função, cujo propósito é enviar os dados do sensor para a Internet e para a página de feed do Pachube:

```
void pachube_out(){
```

Você inicia chamando a função `getTemperatures()` para armazenar as leituras de ambos os sensores:

```
getTemperatures();
```

Então, verifica se o valor atual em `millis()` é menor do que o valor armazenado em `last_connect`. Se afirmativo, então `last_connect` é atualizada com o valor atual de `millis()`. O valor em `last_connect` será atualizado com o valor de `millis()` toda vez que você se desconectar do Pachube. Você utilizará esse valor para verificar quantos milissegundos transcorreram desde sua última conexão.

```
if (millis() < last_connect) last_connect = millis();
```

Em seguida, temos uma instrução `if` que será executada se `request_pause` for verdadeira. Essa variável é definida como `true` apenas se a conexão falhou, ou se você acabou de se desconectar do Pachube.

```
if (request_pause){
```

Dentro dessa instrução `if` você verifica se o valor atual em `last_connect`, subtraído do valor de `millis()`, é maior do que o valor de `interval`. Se afirmativo, isso significa que transcorreu o período de intervalo, e definimos as três flags como `true` ou `false`, de acordo:

```
if ((millis() - last_connect) > interval) {
  ready_to_update = true;
  reading_pachube = false;
  request_pause = false;
}
```

Essas flags dizem à função `pachube_out()` que o intervalo de tempo especificado desde a última conexão, ou tentativa de conexão, já transcorreu e que, portanto, você está pronto para tentar a próxima atualização. Caso você esteja pronto para atualizar, executamos a próxima instrução `if`.

```
if (ready_to_update) {
```

Ela inicia informando ao usuário que você está se conectando:

```
Serial.println("Connecting...");
```

Então, verifica se você conseguiu se conectar com o cliente:

```
if (localClient.connect()) {
```

Se afirmativo, a instrução executa o restante do código, atualizando o feed. Primeiramente, você utiliza um comando `sprintf` para imprimir seus dados formatados em uma string. O comando `sprintf` (string formatada para impressão) é uma forma excelente de reunir elementos distintos de informação em uma string.

```
sprintf(pachube_data,"%d,%d,%d,%d",itempC, itempF, etempC, etempF);
```

Esse comando aceita uma quantidade variável de parâmetros: o primeiro parâmetro é a variável na qual você armazenará os dados formatados (nesse caso, `pachube_data`); o segundo parâmetro especifica o conteúdo da string com marcadores especias (os `%d` da string); do terceiro parâmetro em diante, são declaradas as variáveis que serão incorporadas à string do segundo parâmetro, nas posições dos marcadores especiais. O comando fará a inserção da primeira variável (`itempC`) na string, no ponto em que se encontra o primeiro `%d`; da segunda variável (`itempF`), no segundo `%d` e assim por diante. Os quatro marcadores estão separados por vírgulas, de modo que os números ficarão separados por vírgulas na string final. Assim, se os valores das variáveis fossem:

```
itempC   25
itempF   77
etempC   14
tempF    52
```

Então, depois de executar o comando `sprintf`, o conteúdo de `pachube_data` seria:

```
"25,77,14,52"
```

Se o comando `sprintf` for:

```
sprintf(pachube_data,"Internal Temps: %d,%d External Temps: %d,%d",itempC, itempF, etempC, etempF);
```

Então, `pachube_data` armazenará:

```
"Internal Temps: 25,77 External Temps: 14,52"
```

Como você pode ver, o comando `sprintf` é uma poderosa ferramenta, capaz de converter longas combinações de strings e números em uma única string.

Em seguida, você informa ao usuário quais dados estão sendo enviados:

```
Serial.print("Sending: ");
Serial.println(pachube_data);
```

Então, verifica o comprimento da string em `pachube_data` utilizando a função `strlen()`:

```
content_length = strlen(pachube_data);
```

E informa ao usuário que você está prestes a atualizar o feed:

```
Serial.println("Updating.");
```

Depois, você imprime os dados em `localClient`. Para fazê-lo, você constrói uma string que é enviada ao URL de seu feed. A primeira parte dessa string é o URL do feed, incluindo seu ID, seguido pela chave da API:

```
localClient.print("PUT /v1/feeds/");
localClient.print(SHARE_FEED_ID);
localClient.print(".csv HTTP/1.1\nHost: api.pachube.com\nX-PachubeApiKey: ");
localClient.print(PACHUBE_API_KEY);
```

Em seguida, você envia o nome do agente de usuário. Este é um comando HTTP que geralmente identifica qual software está sendo utilizado como parte da comunicação cliente-servidor. O campo User-Agent do cabeçalho de requisição contém informações sobre o agente de usuário que origina a solicitação. Mesmo não sendo absolutamente necessário, é considerado adequado identificar o código do cliente.

```
localClient.print("\nUser-Agent: Beginning Arduino - Project 47");
```

Depois, você diz ao cliente qual o tipo do conteúdo e seu comprimento. Em nosso caso, temos um arquivo de texto do tipo `.csv`.

```
localClient.print("\nContent-Type: text/csv\nContent-Length: ");
localClient.print(content_length);
localClient.print("\nConnection: close\n\n");
```

Por fim, temos a string com os valores do sensor, separados por vírgula:

```
localClient.print(pachube_data);
localClient.print("\n");
```

As flags são todas redefinidas com seus valores originais:

```
ready_to_update = false;
reading_pachube = true;
request_pause = false;
interval = UPDATE_INTERVAL;
}
```

Se você foi incapaz de se conectar, avisa o usuário, define as flags apropriadas e reinicia a conexão Ethernet:

```
else {
  Serial.print("connection failed!");
  ready_to_update = false;
  reading_pachube = false;
  request_pause = true;
  last_connect = millis();
  interval = RESET_INTERVAL;
```

```
    setupEthernet();
  }
```

Depois, temos uma instrução `while` para verificar se `reading_pachube` é verdadeiro; em caso positivo, ela verifica se `localClient` tem dados disponíveis (por exemplo, se uma resposta foi enviada a partir do Pachube) e, se houver dados, chama a função `checkForResponse()`. Se `localClient` não estiver conectado, nos desconectamos do Pachube executando a função `disconnect_pachube()`:

```
    while (reading_pachube) {
      while (localClient.available()) {
        checkForResponse();
      }
      if (!localClient.connected()) {
        disconnect_pachube();
      }
    }
  }
```

A função `disconnect_pachube()` informa ao usuário que você está se desconectando, interrompe `localClient`, define as flags com seus valores originais e reinicializa o shield Ethernet:

```
  void disconnect_pachube() {
    Serial.println("disconnecting.\n===============\n\n");
    localClient.stop();
    ready_to_update = false;
    reading_pachube = false;
    request_pause = true;
    last_connect = millis();
    resetEthernetShield();
  }
```

A função `checkForResponse()` tem o objetivo de verificar se o cliente (nesse caso, o Pachube) enviou um comando `"200 OK"`, sinalizando que o envio dos dados teve êxito. Então, ela verifica se você está no fim da string sendo enviada (`\n`) e, se afirmativo, limpa o buffer, deixando-o pronto para a próxima operação.

```
  void checkForResponse() {
```

O comando `read()` é utilizado para receber um byte do cliente e armazená-lo em `c`:

```
  char c = localClient.read();
```

O byte recebido é armazenado no array `buff`:

```
  buff[pointer] = c;
```

Se você não tiver ultrapassado 64 bytes em comprimento (o tamanho do array), o índice será incrementado:

```
if (pointer < 64) pointer++;
```

O valor em `c` é verificado, para descobrir se você recebeu o fim da string:

```
if (c == '\n') {
```

Se afirmativo, você utiliza o comando `strstr` para verificar se a sequência `"200 OK"` surge em algum ponto da string e retorna um ponteiro para sua localização, armazenado em `found`:

```
found = strstr(buff, "200 OK");
```

O comando `strstr` encontra uma sub-string dentro de outra string e requer dois parâmetros: o primeiro é a string que você está verificando, e o segundo é a string que você deseja localizar. Se a substring for encontrada, o comando retorna sua localização, caso contrário, retorna um ponteiro nulo.

Então, reinicializamos e limpamos o buffer:

```
buff[pointer]=0;
clean_buffer();
```

Em seguida, temos a função final para obter as temperaturas dos sensores DS18B20:

```
void getTemperatures() {
  sensors.requestTemperatures();
  itempC = sensors.getTempC(insideThermometer);
  itempF = DallasTemperature::toFahrenheit(itempC);
  etempC = sensors.getTempC(outsideThermometer);
  etempF = DallasTemperature::toFahrenheit(etempC);
}
```

Depois de definir todas as suas funções, você alcança a rotina de inicialização:

```
void setup()
```

A primeira coisa que ela faz é iniciar a comunicação serial com taxa de transmissão de 57.600 bauds:

```
Serial.begin(57600);
```

Então, chama a função `setupEthernet()`:

```
setupEthernet();
```

Depois, iniciamos os sensores 1-Wire e definimos sua resolução:

```
sensors.begin();
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

Agora, basta que o loop principal não faça nada além de chamar a função `pachube_out` repetidas vezes:

```
void loop() {
  pachube_out();
}
```

Agora que você conhece os métodos básicos para enviar dados de sensores ao Pachube (onde eles poderão ser armazenados e visualizados), será relativamente fácil modificar o código e adicionar mais sensores de outros tipos de dados.

Até aqui, você aprendeu a enviar dados pela Ethernet a um navegador web na rede, para um navegador web pela Internet e para o serviço de armazenamento de dados e criação de gráficos do Pachube. A seguir, você verá como fazer o Arduino lhe enviar um alerta por e-mail quando a temperatura atingir níveis altos ou baixos demais.

EXERCÍCIO

Na linguagem de programação C, você pode utilizar `%f` com `sprintf` para imprimir `floats`. Todavia, isso não funciona com a versão C do Arduino. Nela, apenas inteiros são impressos. Sendo assim, modifique o código para que as temperaturas sejam enviadas como números em ponto flutuante. Dica: você já converteu manualmente dígitos em strings em outro ponto do livro.

Projeto 48 – Sistema de alerta por e-mail

Agora você verá um método diferente de envio de dados. Neste projeto, você fará com que o Arduino e o shield Ethernet enviem um e-mail quando as temperaturas estiverem altas ou baixas demais. Este projeto tem como objetivo mostrar-lhe o básico do envio de e-mails por meio do shield Ethernet. Você utilizará o mesmo circuito que acabamos de ver, mas, dessa vez, com apenas um dos sensores de temperatura.

Digite o código

Digite o código da listagem 17.3. Você terá de obter o endereço IP de seu servidor SMTP de e-mail. Para isso, abra uma janela de terminal (janela de comando, console, ou qualquer outro nome que ela tenha em seu sistema) e digite `ping`, seguido pelo endereço do qual você deseja obter o IP. Em outras palavras, se você quiser saber o endereço IP do servidor SMTP do Hotmail, em *smtp.live.com*, digite:

```
ping smtp.live.com
```

E receberá uma resposta semelhante a:

```
PING smtp.hot.glbdns.microsoft.com (65.55.162.200): 56 data bytes
```

Isso mostra que o endereço IP é 65.55.162.200. Faça isso para o servidor SMTP de seu serviço de e-mail e digite essa informação na seção do código relativa ao servidor.

Caso seu servidor SMTP exija autenticação, você terá de obter a versão em Base 64 de seu nome de usuário e senha. Há muitos sites que fazem isso para você, como:

```
www.motobit.com/util/base64-decoder-encoder.asp
```

Digite e criptografe seu nome de usuário em Base 64. Depois, faça o mesmo com sua senha. Copie e cole os resultados na seção relevante do código. Da mesma forma, altere as seções `FROM` e `TO`, colocando seu próprio endereço de e-mail e o endereço de e-mail do destinatário.

Listagem 17.3 – Código para o projeto 48

```
// Projeto 48 – Sistema de alerta por e-mail

#include <Ethernet.h>
#include <SPI.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define time 1000
#define emailInterval 60
#define HighThreshold 40  // Temperatura mais alta permitida
#define LowThreshold 10 // Temperatura mais baixa

// Fio de dados é plugado ao pino 3 do Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float tempC, tempF;
char message1[35], message2[35];
char subject[] = "ARDUINO: TEMPERATURE ALERT!!\0";
unsigned long lastMessage;

// Prepara uma instância de OneWire (chamada oneWire) para se comunicar com qualquer
// dispositivo 1-Wire
OneWire oneWire(ONE_WIRE_BUS);

// Passa o endereço da instância oneWire ao Dallas Temperature.
DallasTemperature sensors(&oneWire);

// arrays para armazenar os endereços dos dispositivos
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };

byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };
byte ip[] = { 192, 168, 0, 105 };
byte server[] = { 62, 234, 219, 95 };  // Endereço do servidor de e-mail. Altere esses dados
                                       // para correspodnerem ao IP de seu próprio servidor.
```

```
Client client(server, 25);

void sendEmail(char subject[], char message1[], char message2[], float temp) {
  Serial.println("connecting...");

  if (client.connect()) {
    Serial.println("connected");
    client.println("EHLO MYSERVER");  delay(time); // faz o log in
    client.println("AUTH LOGIN"); delay(time); // autoriza
    // digite seu nome de usuário aqui
    client.println("caFzLmNvbQaWNZXGluZWVsZWN0cm9uNAZW2FsydGhzd3");  delay(time);
    // e a senha aqui
    client.println("ZnZJh4TYZ2ds");  delay(time);
    client.println("MAIL FROM:<alert@bobsmith.org>");   delay(time);
    client.println("RCPT TO:<fred@bloggs.com>");   delay(time);
    client.println("DATA");       delay(time);
    client.println("From: <alert@bobsmith.org>");  delay(time);
    client.println("To: <fred@bloggs.com>");   delay(time);
    client.print("SUBJECT: ");
    client.println(subject);   delay(time);
    client.println();    delay(time);
    client.println(message1);    delay(time);
    client.println(message2);    delay(time);
    client.print("Temperature: ");
    client.println(temp);    delay(time);
    client.println(".");   delay(time);
    client.println("QUIT");    delay(time);
    Serial.println("Email sent.");
    lastMessage=millis();
  } else {
    Serial.println("connection failed");
  }
}

void checkEmail() { // verifica se há dados disponíveis no cliente
  while (client.available()) {
    char c = client.read();
    Serial.print(c);
  }

  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
  }
}
```

```
// função para obter a temperatura de um dispositivo
void getTemperature(DeviceAddress deviceAddress) {
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}

void setup() {
  lastMessage = 0;
  Ethernet.begin(mac, ip);
  Serial.begin(9600);

  // Inicializa a biblioteca sensors
  sensors.begin();
  // define a resolução
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);

  delay(1000);
}

void loop() {
  sensors.requestTemperatures();
  getTemperature(insideThermometer);
  Serial.println(tempC);

  // Está quente demais?
  if (tempC >= HighThreshold && (millis()>(lastMessage+(emailInterval*1000)))) {
    Serial.println("High Threshhold Exceeded");
    char message1[] = "Temperature Sensor\0";
    char message2[] = "High Threshold Exceeded\0";
    sendEmail(subject, message1, message2, tempC);
  } // Frio demais?
  else if (tempC<= LowThreshold && (millis()>(lastMessage+(emailInterval*1000))))
    Serial.println("Low Threshhold Exceeded");
    char message1[] = "Temperature Sensor\0";
    char message2[] = "Low Threshold Exceeded\0";
    sendEmail(subject, message1, message2, tempC);
  }
  if (client.available()) {checkEmail();}
}
```

Faça o upload do código e depois abra o monitor serial. Você verá que ele exibe repetidas vezes a temperatura do primeiro sensor. Se a temperatura descer abaixo do valor de `LowThreshold`, o monitor serial exibirá `"Low Threshold Exceeded"` (limiar mínimo excedido) e enviará o alerta correspondente por e-mail. Se a temperatura ultrapassar o valor de `HighThreshold`, o monitor exibirá a mensagem `"High Threshold Exceeded"` (limiar máximo excedido) e enviará o alerta apropriado por e-mail, indicando uma situação de temperatura elevada.

Você pode testar essa funcionalidade definindo o limiar de temperatura máxima um pouco acima da temperatura ambiente e segurando o sensor em suas mãos, até que a temperatura ultrapasse esse limite. Isso deve disparar o alerta.

Note que, durante os primeiros 60 segundos, o sistema ignorará situações de alerta. Os alertas passarão a ser enviados apenas depois de transcorridos 60 segundos. Se os limiares forem atingidos, o sistema de alerta continuará enviando e-mails até que a temperatura desça abaixo dos níveis aceitáveis. E-mails serão enviados a cada intervalo de segundos definido em `emailInterval`, enquanto os limites estiverem ultrapassados. Você pode ajustar esse intervalo com suas próprias configurações.

Depois que um e-mail tiver sido enviado, o sistema aguardará até que o cliente envie uma notificação de que recebeu a mensagem e, então, exibirá a resposta. Você pode utilizar esses dados para depurar o sistema, caso a implementação não funcione como previsto.

Projeto 48 – Sistema de alerta por e-mail – Análise do código

Primeiramente, incluímos as bibliotecas:

```
#include <Ethernet.h>
#include <SPI.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Depois, você define o intervalo, em milissegundos, para envio de dados ao servidor:

```
#define time 1000
```

Seguido por um tempo, em segundos, entre o envio dos e-mails.

```
#define emailInterval 60
```

Então, você tem de definir os níveis de temperatura máxima e mínima que provocarão um alerta:

```
#define HighThreshold 40  // Temperatura mais alta permitida
#define LowThreshold 10   // Temperatura mais baixa
```

Depois, você define o pino e a precisão dos sensores:

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

E as variáveis `float` para armazenar as temperaturas:

```
float tempC, tempF;
```

Depois, definimos dois arrays de caracteres, que armazenarão a mensagem no e-mail:

```
char message1[35], message2[35];
```

Além de outro array de caracteres, que declaramos e inicializamos, para armazenar o assunto do e-mail:

```
char subject[] = "ARDUINO: TEMPERATURE ALERT!!\0";
```

Como você não deseja bombardear o usuário com mensagens seguidas quando os limites tiverem sido excedidos, deve armazenar o tempo em que o último e-mail foi enviado. Esse dado será armazenado em um inteiro não sinalizado, `lastMessage`:

```
unsigned long lastMessage;
```

Depois, preparamos as instâncias do sensor, acompanhadas do endereço dele:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
```

Definimos também os endereços MAC e IP do shield Ethernet:

```
byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };
byte ip[] = { 192,168,0, 105 };
```

Depois, você define o endereço IP de seu servidor SMTP de e-mails. Aqui, você deve incluir o endereço apropriado ao seu caso, ou o código não funcionará:

```
byte server[] = { 62, 234, 219, 95 };
```

Uma instância `client` é criada, e você passa a ela o endereço do servidor, além do número de porta 25. Caso seu servidor SMTP esteja utilizando uma porta diferente, altere esse dado como necessário:

```
Client client(server, 25);
```

Em seguida, você tem a primeira de suas funções, responsável por enviar o e-mail ao servidor. Ela requer quatro parâmetros: o assunto do e-mail, a primeira linha da mensagem, a segunda linha da mensagem e, por fim, a temperatura.

```
void sendEmail(char subject[], char message1[], char message2[], float temp) {
```

Você informa ao usuário que está tentando se conectar:

```
Serial.println("connecting...");
```

Depois, verifica se o cliente se conectou. Se afirmativo, executamos o código do bloco `if`:

```
if (client.connect()) {
```

Primeiramente, o usuário é informado de que você se conectou ao cliente. O cliente, nesse caso, é seu servidor SMTP de e-mail:

```
Serial.println("connected");
```

Depois, você envia comandos ao servidor praticamente da mesma forma que no projeto 46. Primeiramente, você deve se apresentar ao servidor SMTP. Isso é feito com um comando EHLO acompanhado dos detalhes do servidor. Depois de cada comando, você deve aguardar um pouco para que ele seja processado. Em meu caso, verifiquei que mil milissegundos foram suficientes; você pode ter de alterar esse número de acordo com suas necessidades:

```
client.println("EHLO MYSERVER");  delay(time); // faz o log in
```

Isso é como um procedimento de handshake[1] entre o servidor e o cliente, pelo qual eles se apresentam um ao outro. Em seguida, você tem de autorizar a conexão. Se o seu servidor SMTP não exigir autorização, você pode comentar essa linha e as linhas que escrevem o nome de usuário e a senha.

```
client.println("AUTH LOGIN"); delay(time);    // autoriza
```

Às vezes o servidor exige um login não criptografado. Nesses casos, você teria de enviar `AUTH PLAIN` acompanhado do nome de usuário e da senha em texto simples.

Em seguida, os dados criptografados em Base 64 devem ser enviados ao servidor:

```
client.println("caFzLmNvbQaWNZXGluZWVsZWN0cm9uNAZW2FsydGhzd3");  delay(time);
client.println("ZnZJh4TYZ2ds");  delay(time);
```

Então, você tem de dizer ao servidor de quem está vindo o e-mail, e para quem ele está sendo enviado:

```
client.println("MAIL FROM:<alert@bobsmith.org>");      delay(time);
client.println("RCPT TO:<fred@bloggs.com>");           delay(time);
```

Essas informações devem ser alteradas para que correspondam ao seu próprio endereço de e-mail e ao e-mail do destinatário. A maioria dos servidores SMTP apenas permitirá que você envie e-mails utilizando um endereço de e-mail do mesmo domínio (por exemplo, você não pode enviar um e-mail de uma conta no Hotmail utilizando um servidor do Yahoo).

Em seguida, temos o comando `DATA` para informar ao servidor que o que vem a seguir são os dados do e-mail, ou seja, aquilo que será visível ao destinatário.

```
client.println("DATA");    delay(time);
```

Você deseja que o destinatário veja informações de remetente e destinatário, por isso incluímos novamente essas informações em benefício dele.

```
client.println("From: <alert@bobsmith.org>");          delay(time);
client.println("To: <fred@bloggs.com>");               delay(time);
```

1 N.T.: Handshake, ou aperto de mão, é o processo pelo qual duas máquinas afirmam uma à outra que se reconheceram e estão prontas para iniciar a comunicação. O handshake é utilizado em protocolos de comunicação, tais como: FTP, TCP, HTTP, SMB, SMTP, POP3 etc. (fonte: Wikipédia).

Em seguida, você envia o assunto do e-mail: a palavra “`SUBJECT:`” seguida pelo assunto informado à função:

```
client.print("SUBJECT: ");
client.println(subject);   delay(time);
```

Antes do corpo do e-mail, você deve enviar uma linha em branco:

```
client.println();   delay(time);
```

Seguida pelas duas linhas da mensagem passada à função:

```
client.println(message1);  delay(time);
client.println(message2);  delay(time);
```

Então, você inclui a temperatura:

```
client.print("Temperature: ");
client.println(temp);    delay(time);
```

Todos os e-mails devem ter um ponto (.) ao final, em uma linha própria, para dizer ao servidor que você terminou:

```
client.println(".");  delay(time);
```

Então, você envia um comando `QUIT` para se desconectar do servidor:

```
client.println("QUIT");    delay(time);
```

Por fim, informamos ao usuário que o e-mail foi enviado:

```
Serial.println("Email sent.");
```

Em seguida, armazenamos o valor atual de `millis()` em `lastMessage`, uma vez que você utilizará essa informação futuramente para verificar se transcorreu o intervalo de tempo especificado entre as operações de envio de mensagens.

```
lastMessage=millis();
```

Caso a conexão com o cliente não seja bem sucedida, o e-mail não será enviado e o usuário será informado:

```
} else {
  Serial.println("connection failed");
}
```

Depois, temos a função que lê a resposta do cliente:

```
void checkEmail() { // verifique se há dados disponíveis no cliente
```

Enquanto houver dados disponíveis para leitura no cliente:

```
while (client.available()) {
```

Você armazena esse byte em `c`:

```
char c = client.read();
```

E o imprime na janela do monitor serial:

```
Serial.print(c);
```

Se o cliente NÃO estiver conectado:

```
if (!client.connected()) {
```

Então informamos ao usuário que o sistema está se desconectando, e o cliente conectado é interrompido:

```
Serial.println();
Serial.println("disconnecting.");
client.stop();
```

Em seguida, temos a função que você utilizou antes para obter as temperaturas do sensor 1-Wire:

```
void getTemperature(DeviceAddress deviceAddress) {
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}
```

Seguida por uma rotina de inicialização que simplesmente configura a Ethernet e os sensores:

```
void setup() {
  Ethernet.begin(mac, ip);
  Serial.begin(9600);

  // Inicializa a biblioteca sensors
  sensors.begin();
  // define a resolução
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);

  delay(1000);
}
```

Finalmente, temos o loop principal do programa:

```
void loop()
```

Você inicia solicitando as temperaturas da biblioteca Dallas Temperature:

```
sensors.requestTemperatures();
```

Depois, chama sua função `getTemperature`, informando a ela o endereço do sensor:

```
getTemperature(insideThermometer);
```

A temperatura é, então, exibida na janela do monitor serial:

```
Serial.println(tempC);
```

Em seguida, você verifica se essa temperatura atingiu ou excedeu o limiar máximo. Se afirmativo, então o e-mail apropriado é enviado. Entretanto, você deseja enviar apenas um e-mail a cada intervalo de `(emailInterval*1000)` segundos, por isso também verifica se `millis()` é maior do que o valor da última vez que a mensagem de e-mail foi enviada `(lastMessage)` somado ao tempo de intervalo. Se afirmativo, executamos o código.

```
if (tempC >= HighThreshold && (millis()>(lastMessage+(emailInterval*1000)))) {
```

Informamos o usuário e enviamos as duas linhas que formam a mensagem de e-mail:

```
Serial.println("High Threshhold Exceeded");
char message1[] = "Temperature Sensor\0";
char message2[] = "High Threshold Exceeded\0";
```

Depois, chamamos a função `sendEmail`, passando a ela os parâmetros que correspondem ao assunto, às linhas um e dois da mensagem e à temperatura atual:

```
    sendEmail(subject, message1, message2, tempC);
```

Se o limiar de temperatura máxima não foi atingido, você verifica se a temperatura está abaixo do limiar mínimo. Caso afirmativo, execute o mesmo procedimento com a mensagem apropriada.

```
else if (tempC<= LowThreshold && (millis()>(lastMessage+(emailInterval*1000))))
  Serial.println("Low Threshhold Exceeded");
  char message1[] = "Temperature Sensor\0";
  char message2[] = "Low Threshold Exceeded\0";
  sendEmail(subject, message1, message2, tempC);
}
```

Por fim, você verifica se existem dados prontos para serem recebidos de volta do cliente (depois do envio de um e-mail) e exibe os resultados:

```
if (client.available()) {checkEmail();}
```

Esses dados são úteis para depuração.

Este projeto apresentou-lhe informações básicas necessárias para envio de um e-mail, a partir do Arduino, usando o shield Ethernet. Essa funcionalidade poderia ser utilizada para enviar alertas ou notificar a ocorrência de uma ação, como quando se detecta a entrada de uma pessoa em um quarto ou a abertura de uma caixa. O sistema também pode executar outras ações, como abrir uma janela, se a temperatura no quarto ficar quente demais, ou completar o nível de um tanque de peixes, se o nível da água descer muito.

Em seguida, você aprenderá a enviar dados de um Arduino para o Twitter.

Projeto 49 – Twitterbot

Novamente, você utilizará o circuito com dois sensores de temperatura. Dessa vez, você enviará ao Twitter atualizações regulares referentes ao status dos dois sensores. Para isso, criaremos um sistema simples de verificação do status dos sensores conectados ao Arduino.

O Twitter é um serviço de microblogging que permite o envio de posts em miniatura, ou "tweets", de 140 caracteres de comprimento. Os tweets ficam disponíveis publicamente a qualquer pessoa que efetue uma busca ou àqueles que escolham assinar (ou seguir) seu blog. Trata-se de um serviço incrivelmente popular, e que pode ser acessado a partir de qualquer navegador web ou de um dos muitos clientes disponíveis do Twitter, incluindo aplicativos de dispositivos móveis. Isso o torna ideal para enviar pequenos textos de informação, que podem ser verificados enquanto você faz outras coisas.

Você terá de acessar o Twitter.com e criar uma nova conta. Recomendo a criação de uma conta que será utilizada exclusivamente para tweets referentes ao seu Arduino.

A partir de 31 de agosto de 2010, o Twitter alterou sua política no que se refere ao acesso ao site utilizando aplicativos de terceiros. Um método de autenticação, conhecido como OAuth, agora é utilizado, tornando muito mais difícil twittar diretamente a partir de um Arduino; antes dessa alteração, esse processo era muito mais fácil. Twittar, neste momento, só é possível utilizando um recurso externo. Em outras palavras, é necessário enviar o tweet para um site, ou proxy, que twittará em seu nome, utilizando o token OAuth (código de acesso). A biblioteca atual do Twitter utiliza esse método.

Assim que você tiver criado sua conta, digite o código da próxima seção.

Digite o código

No momento da redação deste livro, as bibliotecas Ethernet, das quais depende atualmente a biblioteca Twitter, funcionam apenas com o IDE versão 0018 do Arduino. Você terá, portanto, de visitar o site do Arduino, navegar até a página de download e obter o IDE adequado. Certifique-se de executar e fazer o upload do código apenas com essa versão do IDE, mantendo-a separada de sua versão mais atual. As bibliotecas `EthernetDNS` e `EthernetDHCP` que utilizamos com a biblioteca do Twitter podem ser encontradas em *http://gkaindl.com/software/arduino-ethernet*. Assim que essa biblioteca tiver sido atualizada para funcionar com o IDE mais recente, você poderá utilizá-lo.

Antes de fazer o upload do código, você deve ter um token para a conta no Twitter. A biblioteca que você está utilizando foi criada pelo usuário NeoCat, e utiliza o site dele como proxy para enviar o tweet. Isso significa que você tem primeiro de obter um token (versão criptografada de seu nome de usuário e senha) para acessar o site

do Twitter. Para isso, visite o site de NeoCat, em *http://arduino-tweet.appspot.com*, e clique no link "Step 1" (passo 1) para obter o token. Copie e cole esses dados na seção de token do código.

Note que, como você está utilizando um proxy e tem de transmitir seu nome de usuário e senha para obter o token, é aconselhável criar uma nova conta no Twitter e mantê-la anônima (ou seja, não adicione nomes ou endereços de e-mail ao perfil dessa conta). Estou certo de que é perfeitamente seguro utilizar essa biblioteca com sua própria conta se você preferir, mas é melhor prevenir que remediar.

Depois, clique no link "Step 2" (passo 2) e obtenha as duas bibliotecas das quais depende o código. Instale-as na pasta `libraries` da versão 0018 do IDE do Arduino, cujo download e instalação já vimos. Será necessário reiniciar o IDE antes que você possa utilizar essas bibliotecas. A biblioteca Twitter vem também com alguns exemplos que você pode conferir. Se você deseja ler mais sobre essa biblioteca, pode encontrá-la no Playground do Arduino, em *www.arduino.cc/playground/Code/TwitterLibrary*.

Assim que você tiver seu token e suas bibliotecas instaladas, digite e faça o upload do código da listagem 17.4.

Listagem 17.4 – Código para o projeto 49

```
// Projeto 49 - Twitterbot

#include <Ethernet.h>
#include <EthernetDHCP.h>
#include <EthernetDNS.h>
#include <Twitter.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Fio de dados é plugado ao pino 3 do Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float itempC, itempF, etempC, etempF;
boolean firstTweet = true;
// Prepara uma instância de OneWire (chamada oneWire) para se comunicar com qualquer
// dispositivo 1-Wire (não apenas com CIs de temperatura Maxim/Dallas)
OneWire oneWire(ONE_WIRE_BUS);

// Passa o endereço da instância oneWire ao Dallas Temperature.
DallasTemperature sensors(&oneWire);

// arrays para armazenar os endereços dos dispositivos
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
```

```
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};

byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };

// Seu token para twittar (obtenha essa informação em http://arduino-tweet.appspot.com/)
Twitter twitter("608048201-CxY1yQi8ezhvjz60ZVfPHVdzIHbMOD1h2gvoaAIx");

unsigned long interval = 600000; // 10 minutos
unsigned long lastTime; // tempo transcorrido desde o último tweet

// Mensagem para postar
char message[140], serialString[60];

// função para obter a temperatura de um dispositivo
void getTemperatures() {
  itempC = sensors.getTempC(insideThermometer);
  itempF = DallasTemperature::toFahrenheit(itempC);
  etempC = sensors.getTempC(outsideThermometer);
  etempF = DallasTemperature::toFahrenheit(etempC);
}

void tweet(char msg[]) {
  Serial.println("connecting ...");
  if (twitter.post(msg)) {
    int status = twitter.wait();
    if (status == 200) {
      Serial.println("OK. Tweet sent.");
      Serial.println();
      lastTime = millis();
      firstTweet = false;
    } else {
      Serial.print("failed : code ");
      Serial.println(status);
    }
  } else {
    Serial.println("connection failed.");
  }
}

void setup() {
  EthernetDHCP.begin(mac);
  Serial.begin(9600);
  sensors.begin();
  // define a resolução
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
  sensors.requestTemperatures()

  getTemperatures();
  // compila as strings a serem twittadas
```

```
while (firstTweet) {
sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F).  Tweeted from Arduino. %ld",
  int(itempC), int(itempF), int(etempC), int(etempF), millis());        tweet(message);
  }
}

void loop() {
  EthernetDHCP.maintain();
  sensors.requestTemperatures();
  // compila a string a ser impressa no monitor serial
  sprintf(serialString, "Internal Temp: %d C  %d F. External Temp: %d C %d F", int(itempC),
    int(itempF), int(etempC), int(etempF));
  delay(500);
  Serial.println(serialString);
  Serial.println();

  if (millis() >= (lastTime + interval)) {
    // compila a string a ser twittada
    sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from Arduino.
%ld", int(itempC), int(itempF), int(etempC), int(etempF), millis());   tweet(message);
  }
  delay(10000); // 10 segundos
}
```

Depois do upload do código para seu Arduino, abra a janela do monitor serial. O Arduino tentará se conectar ao Twitter (na verdade, ao site de NeoCat) e enviar o tweet. Se isso der certo, a saída no monitor serial será semelhante a esta:

```
connecting ...
OK. Tweet sent.

Internal Temp: 26 C  79 F. External Temp: 26 C 79 F

Internal Temp: 26 C  79 F. External Temp: 26 C 79 F

Internal Temp: 26 C  79 F. External Temp: 26 C 79 F
```

Quando o programa for executado pela primeira vez, ele obterá a temperatura e continuará tentando se conectar ao Twitter na rotina de inicialização, antes de avançar para o loop principal. Esse processo continuará até que a conexão ocorra. Se o programa não conseguir se conectar, você receberá um código de falha 403, ou uma mensagem de falha de conexão. Se o tweet der certo, o código não twittará novamente, até que o período de intervalo tenha transcorrido. Por padrão, esse intervalo é definido como dez minutos, mas você pode alterá-lo. O Twitter limita um máximo de 350 solicitações por hora, por isso não exagere. Agora você pode acessar o site do Twitter e visualizar sua conta onde quer que esteja, conferindo as leituras de temperatura.

Vejamos como esse código funciona.

Projeto 49 – Twitterbot – Análise do código

O programa inicia incluindo as bibliotecas relevantes:

```
#include <Ethernet.h>
#include <EthernetDHCP.h>
#include <EthernetDNS.h>
#include <Twitter.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

A biblioteca Twitter necessita de três bibliotecas Ethernet para funcionar, por isso todas elas foram incluídas. Em seguida, apresentamos as definições para os sensores:

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

Você cria quatro `floats` para as temperaturas internas e externas, tanto em Celsius, quanto em Fahrenheit:

```
float itempC, itempF, etempC, etempF;
```

Na primeira tentativa do programa em fazer um tweet, você deseja que ele continue tentando até que consiga se conectar e enviar a mensagem. Portanto, criamos uma variável booleana, definida como `true`, para que você saiba se o primeiro tweet ainda está para ser feito:

```
boolean firstTweet = true;
```

Assim como antes, você cria instâncias para os sensores 1-Wire e de temperatura, além dos endereços para os dois sensores:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};
```

Você dá ao shield Ethernet um endereço MAC:

```
byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };
```

Em seguida, cria uma instância da biblioteca Twitter e passa a ela o token da sua conta:

```
Twitter twitter("608048201-CxY1yQi8ezhvjz60ZVfPHVdzIHbMOD1h2gvoaAIx");
```

Depois, definimos o intervalo entre os tweets:

```
unsigned long interval = 600000; // 10 minutos
```

Além de uma variável para armazenar o tempo transcorrido desde o último tweet:

```
unsigned long lastTime; // tempo transcorrido desde o último tweet
```

Dois arrays de caracteres são criados, para armazenar a mensagem que será twittada e a mensagem que você mostrará na janela do monitor serial.

```
char message[140], serialString[60];
```

Agora, você cria algumas funções. A primeira serve para obter as temperaturas dos dois sensores e armazená-las em suas variáveis.

```
void getTemperatures() {
  itempC = sensors.getTempC(insideThermometer);
  itempF = DallasTemperature::toFahrenheit(itempC);
  etempC = sensors.getTempC(outsideThermometer);
  etempF = DallasTemperature::toFahrenheit(etempC);
}
```

Em seguida, temos a função que fará os tweets para você. Ela requer um parâmetro: o array de caracteres que contém a sua mensagem.

```
void tweet(char msg[]) {
```

Informamos ao usuário que você está tentando se conectar:

```
Serial.println("connecting ...");
```

Depois, você utiliza o método `post()` do objeto `twitter` para enviar a mensagem. Se o post der certo, a função retorna `true`. Se ele falhar ao tentar se conectar, ela retorna `false`.

```
  if (twitter.post(msg)) {
```

Se você conseguir se conectar, verifique o status do post utilizando o método `wait()`. Isso retorna o código de status HTTP presente na resposta vinda do Twitter:

```
int status = twitter.wait();
```

Caso o código de status seja `200`, essa é a forma do código HTTP dizer que está tudo o.k. Em outras palavras, se o tweet foi enviado corretamente, o código dentro do bloco será executado.

```
if (status == 200) {
```

Se a condição for verdadeira, você informa ao usuário:

```
Serial.println("OK. Tweet sent.");
Serial.println();
```

Então, definimos `lastTime` com o valor atual de `millis()`, para que você possa determinar quanto tempo transcorreu desde o último tweet:

```
lastTime = millis();
```

Na primeira vez que um tweet ocorre com sucesso, você deseja que o programa saia do loop `while`, da rotina de inicialização, e avance para o loop principal, por isso você define a flag `firstTweet` como `false`.

```
firstTweet = false;
```

Se o status não for 200, ou seja, se o post tiver falhado, então informamos o ocorrido ao usuário e exibimos o código para depuração:

```
} else {
  Serial.print("failed : code ");
  Serial.println(status);
}
```

Se você não conseguiu nem ao menos se conectar, o usuário será informado desse problema.

```
} else {
  Serial.println("connection failed.");
}
```

Concluídas as funções do usuário, você atinge a rotina de inicialização:

```
void setup()
```

Primeiramente, você inicializa a biblioteca `EthernetDHCP` e passa a ela o endereço MAC:

```
EthernetDHCP.begin(mac);
```

O DHCP (*Dynamic Hosting Configuration Protocol*, ou Protocolo de Configuração de Hospedagem Dinâmica) é um protocolo de configuração automática utilizado em redes IP, que permite ao shield Ethernet receber automaticamente um endereço IP daqueles disponíveis no roteador. Em projetos anteriores, você definiu manualmente o endereço IP; dessa vez, você utilizará a biblioteca `EthernetDHCP` para efetuar essa atribuição automaticamente. A única ressalva é que seu código ficará muito maior.

Em seguida, você inicia as comunicações seriais a uma taxa de 9.600 bauds e define o sensor como antes:

```
Serial.begin(9600);
sensors.begin();
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

Depois, solicitamos as temperaturas, uma vez que você está prestes a utilizá-las:

```
sensors.requestTemperatures()
getTemperatures();
```

Agora, você tenta enviar seu primeiro tweet. Para tanto, o loop `while` seguirá executando enquanto `firstTweet` for `true`:

```
while (firstTweet) {
```

Depois, você utiliza um comando `sprintf` para compilar o tweet no array `message[]`. Você passa a ele quatro conjuntos de temperatura, além do valor de `millis()`. Como `millis` é um número inteiro não sinalizado, você utiliza o especificador `%ld` em `sprintf` para imprimir um inteiro longo.

```
sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from Arduino. %ld",
  int(itempC), int(itempF), int(etempC), int(etempF), millis());
```

O motivo de você adicionar o valor de `millis()` ao final do tweet é que o Twitter não postará uma mensagem idêntica à anterior. Se as temperaturas não sofreram alteração desde o último tweet, a mensagem será a mesma e o Twitter retornará um código de erro. Como você pretende obter atualizações regulares a cada período de intervalo, adicionando o valor de `millis()` ao final você garante que a mensagem será diferente da última enviada. Da mesma forma, certifique-se de que o comprimento do tweet não exceda 140 caracteres; do contrário, você terminará com mensagens estranhas na linha do tempo de seu Twitter.

Agora que compilou sua mensagem, você pode passá-la à função `tweet()`:

```
tweet(message);
```

Em seguida, temos o loop principal, que você atingirá apenas se o primeiro tweet na rotina de inicialização tiver êxito:

```
void loop()
```

Primeiramente, você executa um comando `maintain()` na biblioteca `EthernetDHCP`, o qual mantém ativo e válido o endereço IP atribuído automaticamente.

```
EthernetDHCP.maintain();
```

As temperaturas são atualizadas:

```
sensors.requestTemperatures();
```

Então, você utiliza um comando `sprintf`, compilando a saída para o monitor serial. `sprintf` é mais conveniente do que uma lista completa de comandos `Serial.print()`, por isso vale a pena utilizá-lo, ainda que isso aumente o tamanho de seu código.

```
sprintf(serialString, "Internal Temp: %d C  %d F. External Temp: %d C %d F", int(itempC),
int(itempF), int(etempC), int(etempF));
```

Depois, exibimos a saída no monitor serial após um breve intervalo:

```
delay(500);
Serial.println(serialString);
Serial.println();
```

Em seguida, você verifica se o tempo de intervalo transcorreu desde o último tweet e, se afirmativo, envia mais um tweet. Você calcula o valor de `lastTime + interval`, e verifica se o valor atual em `millis()` é maior que ele (ou seja, se transcorreu o período de intervalo desde o último tweet). Se afirmativo, compilamos uma nova mensagem e efetuamos um novo tweet.

```
if (millis() >= (lastTime + interval)) {
  sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from
    Arduino. %ld", int(itempC), int(itempF), int(etempC), int(etempF), millis());
```

```
    tweet(message);
  }
```

Por fim, você tem uma espera de dez segundos entre as atualizações no monitor serial, para não bombardear o usuário com informações.

```
  delay(10000); // 10 segundos
```

Agora que você aprendeu a enviar tweets a partir de seu Arduino, pode utilizá-lo em todo tipo de projeto. Que tal uma planta que lhe avise quando deve ser aguada? Ou sensores em uma casa para twittar sempre que alguém entra em um quarto; uma campainha que twitta quando alguém chega à porta, ou uma portinha para gatos que lhe avise quando seu gato saiu ou entrou na casa? As possibilidades são infinitas.

Com isso, chegamos ao projeto final de sua jornada. Neste último projeto, utilizaremos o shield Ethernet para ler dados da Internet, em vez de enviá-los.

Projeto 50 – Leitor de RSS meteorológico

O projeto final deste livro utilizará novamente o shield Ethernet, mas, em vez de transmitir dados para um serviço web, você utilizará o shield Ethernet do Arduino para buscar dados da Internet e exibi-los na janela do monitor serial. Para os dados, você utilizará um feed RSS (*Really Simple Syndication*) do site *www.weather.gov*, obtendo dados climáticos referentes a uma área de sua escolha nos Estados Unidos. Este código poderá ser adaptado com facilidade para leitura de um feed RSS meteorológico a partir de qualquer outra fonte, caso você esteja fora dos Estados Unidos.

RSS é um formato web para publicação de informações atualizadas com frequência, como dados meteorológicos, notícias etc. Os dados estão no formato XML (*Extensible Markup Language*, ou Linguagem de Marcação Extensível), conjunto de regras para codificação de documentos de forma a serem lidos por máquinas. Trata-se de um formato simples, não sendo realmente necessário que você compreenda como ele funciona. O Arduino simplesmente procurará tags no código XML nos pontos em que os dados de temperatura, umidade e pressão estão armazenados, retirando essa informação para exibição.

Você utilizará o feed XML da Base Aérea de Edwards, na Califórnia. Caso você queira utilizar um feed diferente, acesse *http://www.weather.gov/xml/current_obs/* e escolha sua área, então procure o endereço completo dos dados XML para esse feed. Ajuste o código de acordo, para mostrar o clima desse local.

Quanto ao hardware, dessa vez você utilizará apenas um shield Ethernet conectado ao Arduino.

Digite o código

Conecte o shield Ethernet ao Arduino (se ele ainda não estiver conectado) e digite o código da listagem 17.5. Agradecemos a Bob S. (Xtalker), participante dos fóruns do Arduino, pelo código.

Listagem 17.5 – Código para o projeto 50

```
// Projeto 50
// Agradecemos a Bob S. pelo código original
// Obtém a observação meteorológica atual da Base Aérea de Edwards no site weather.gov,
// em formato XML

#include <Ethernet.h>
#include <SPI.h>

// Comprimento máximo da string pode ter de ser ajustado, dependendo dos dados a serem extraídos
#define MAX_STRING_LEN  20

// Prepara as variáveis
char tagStr[MAX_STRING_LEN] = "";
char dataStr[MAX_STRING_LEN] = "";
char tmpStr[MAX_STRING_LEN] = "";
char endTag[3] = {'<', '/', '\0'};
int len;
// Flags para diferenciar as tags XML dos elementos do documento (ou seja, dados)
boolean tagFlag = false;
boolean dataFlag = false;

// Variáveis da Ethernet
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = {172,31,24,232};
byte server[] = { 140, 90, 113, 200 }; // www.weather.gov

// Inicializa o cliente ethernet
Client client(server, 80);

void setup() {
  Serial.begin(9600);
  Serial.println("Starting Weather RSS Reader");
  Serial.println("connecting...");
  Ethernet.begin(mac, ip);
  delay(1000);

  if (client.connect()) {
    Serial.println("connected");
```

```
    client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
    client.println();
    delay(2000);
  } else {
    Serial.println("connection failed");
  }
}

void loop() {

  // Lê os dados seriais da web:
  while (client.available()) {
    serialEvent();
  }

  if (!client.connected()) {
    client.stop();

  if (int t=0; t<15; t++) { // o feed é atualizado uma vez a cada 15 minutos
    delay(60000); // 1 minuto
  }

  if (client.connect()) {
      client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
      client.println();
      delay(2000);
    } else {
      Serial.println("Reconnection failed");
    }
  }
}

// Processa cada caractere da web
void serialEvent() {

  // Lê um caractere
  char inChar = client.read();

  if (inChar == '<') {
    addChar(inChar, tmpStr);
    tagFlag = true;
    dataFlag = false;

  } else if (inChar == '>') {
    addChar(inChar, tmpStr);
```

```
    if (tagFlag) {
      strncpy(tagStr, tmpStr, strlen(tmpStr)+1);
    }

    // Limpa o buffer temporário
    clearStr(tmpStr);

    tagFlag = false;
    dataFlag = true;

  } else if (inChar != 10) {
    if (tagFlag) {
      // Adiciona o caractere da tag à string
      addChar(inChar, tmpStr);

      // Verifica a presença da tag de término </XML> e a ignora
      if ( tagFlag && strcmp(tmpStr, endTag) == 0 ) {
        clearStr(tmpStr);
        tagFlag = false;
        dataFlag = false;
      }
    }

    if (dataFlag) {
      // Adiciona caractere de dados à string
      addChar(inChar, dataStr);
    }
  }

  // Se for um LF, processa a linha
  if (inChar == 10 ) {

    // Encontra as tags específicas e imprime os dados
    if (matchTag("<temp_f>")) {
        Serial.print("Temp: ");
        Serial.print(dataStr);
    }
    if (matchTag("<temp_c>")) {
        Serial.print(", TempC: ");
        Serial.print(dataStr);
    }
    if (matchTag("<relative_humidity>")) {
      Serial.print(", Humidity: ");
      Serial.print(dataStr);
    }
```

```
      if (matchTag("<pressure_in>")) {
          Serial.print(", Pressure: ");
          Serial.print(dataStr);
          Serial.println("");
      }

      // Limpa todas as strings
      clearStr(tmpStr);
      clearStr(tagStr);
      clearStr(dataStr);

      // Limpa as flags
      tagFlag = false;
      dataFlag = false;
    }
  }

  // Função para limpar uma string
  void clearStr (char* str) {
    int len = strlen(str);
    for (int c = 0; c < len; c++) {
      str[c] = 0;
    }
  }

  //Função para adicionar um caractere a uma string e verificar seu comprimento
  void addChar (char ch, char* str) {
    char *tagMsg  = "<TRUNCATED_TAG>";
    char *dataMsg = "-TRUNCATED_DATA-";

    // Verifica o tamanho máximo da string, para se certificar de que ela não seja grande demais.
    // Caso a string ultrapasse MAX_STRING_LEN, presume que esses dados não sejam importantes
    // e os substitui por uma mensagem de aviso.
    if (strlen(str) > MAX_STRING_LEN - 2) {
    if (tagFlag) {
    clearStr(tagStr);
        strcpy(tagStr,tagMsg);
      }
      if (dataFlag) {
        clearStr(dataStr);
        strcpy(dataStr,dataMsg);
      }

      // Limpa o buffer temporário e as flags, para interromper o processamento atual
      clearStr(tmpStr);
      tagFlag = false;
```

```
      dataFlag = false;
    } else {
      // Adiciona um caractere à string
      str[strlen(str)] = ch;
    }
  }

  // Função para verificar a tag atual de uma string específica
  boolean matchTag (char* searchTag) {
    if ( strcmp(tagStr, searchTag) == 0 ) {
      return true;
    } else {
      return false;
    }
  }
```

Faça o upload do código e abra o monitor serial. Se tudo estiver funcionando corretamente, você terá uma saída semelhante a:

```
Starting Weather RSS Reader
connecting...
connected
TempF: 60.0, TempC: 15.4, Humidity: 100, Pressure: 29.96
```

A cada 60 segundos, o display será atualizado com dados mais recentes. Vejamos como o código funciona.

Projeto 50 – Leitor de RSS meteorológico – Análise do código

O programa inicia incluindo as bibliotecas necessárias:

```
#include <Ethernet.h>
#include <SPI.h>
```

Depois, você define o comprimento máximo da string de dados:

```
#define MAX_STRING_LEN  20
```

Pode ser necessário elevar esse número, caso você esteja solicitando mais informações do feed. Em seguida, você cria três arrays para armazenar as strings que serão processadas (as quais têm apenas o comprimento definido).

```
char tagStr[MAX_STRING_LEN] = "";
char dataStr[MAX_STRING_LEN] = "";
char tmpStr[MAX_STRING_LEN] = "";
```

Então, você cria outro array para armazenar as possíveis tags de término que podem ser encontradas em um feed XML:

```
char endTag[3] = {'<', '/', '\0'};
```

Depois, você cria uma variável que armazenará, na seção relevante do código, o comprimento da string a ser processada:

```
int len;
```

Agora, você cria duas flags que serão utilizadas para diferenciar as tags XML da informação que vem depois delas, informação essa que você deseja extrair do código XML.

```
boolean tagFlag = false;
boolean dataFlag = false;
```

Em seguida, você configura os endereços MAC e IP do shield Ethernet:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = {172, 31, 24, 232};
```

Seguidos do endereço IP do site *www.weather.gov*:

```
byte server[] = { 140, 90, 113, 200 }; // www.weather.gov
```

Caso você esteja utilizando um site diferente para seu feed meteorológico, altere esse endereço IP de acordo. Em seguida, crie um objeto `client` e passe a ele o endereço do servidor e a porta que você está utilizando:

```
Client client(server, 80);
```

Depois, temos a rotina de inicialização:

```
void setup()
```

Que inicializa a comunicação serial com taxa de 9.600 bauds, para que você possa imprimir dados no monitor serial.

```
Serial.begin(9600);
```

Você informa o nome do programa e sua tentativa de conexão:

```
Serial.println("Starting Weather RSS Reader");
Serial.println("connecting...");
```

Iniciamos as comunicações Ethernet passando os endereços MAC e IP de seu dispositivo, seguidos de um breve intervalo para permitir a conexão:

```
Ethernet.begin(mac, ip);
delay(1000);
```

Em seguida, você verifica se a conexão com o cliente (o site *www.weather.gov*) ocorreu com êxito:

```
if (client.connect()) {
```

Se afirmativo, você informa ao usuário:

```
Serial.println("connected");
```

E executa um comando HTML GET, para acessar os dados XML a partir do subdiretório que armazena o feed relevante, seguido de um intervalo para permitir a comunicação.

```
client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
client.println();
delay(2000);
```

Se a conexão não foi feita, você informa o usuário sobre a falha na conexão:

```
  } else {
    Serial.println("connection failed");
  }
}
```

Depois, temos o loop principal:

```
void loop() {
```

Como você executou um comando GET no loop de inicialização, o buffer serial deve apresentar o conteúdo do feed XML retornado do servidor. Assim, enquanto houver dados disponíveis:

```
while (client.available()) {
```

A função `serialEvent()` será chamada.

Essa função será explicada em breve. Se uma conexão não foi feita:

```
if (!client.connected()) {
```

Interrompemos a conexão com o cliente:

```
client.stop();
```

Depois, você espera 15 minutos antes de tentar outra conexão. O feed de dados é atualizado no máximo uma vez a cada 15 minutos, por isso é inútil atualizar a informação mais rápido que isso:

```
if (int t=0; t<15; t++) {// o feed é atualizado uma vez a cada 15 minutos
  delay(60000); // 1 minuto
}
```

Se conseguiu se conectar ao cliente:

```
if (client.connect()) {
```

Você realiza outro comando GET para obter os dados do feed XML mais recente:

```
client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
client.println();
delay(2000);
```

Se a conexão falhar, você informa ao usuário:

```
} else {
  Serial.println("Reconnection failed");
}
```

Em seguida, temos a função `serialEvent()`. Seu propósito é ler os dados do feed XML e processá-los de acordo com o que for encontrado:

```
void serialEvent() {
```

A função inicia lendo o primeiro caractere e armazenando-o em `inChar`:

```
char inChar = client.read();
```

Agora, você deve verificar o caractere e decidir se ele é uma tag ou se representa dados. Se for uma tag, então definimos `tagFlag` como `true`. Se o caractere representa dados, `dataFlag` é definida como `true`. A outra flag é sempre definida como `false`.

Os dados brutos do feed têm a seguinte apresentação:

```
<current_observation version="1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.weather.gov/view/current_observation.xsd">
  <credit>NOAA's National Weather Service</credit>
  <credit_URL>http://weather.gov/</credit_URL>
  <image><url>http://weather.gov/images/xml_logo.gif</url><title>NOAA's National
Weather Service</title><link>http://weather.gov</link></image>
  <suggested_pickup>15 minutes after the hour</suggested_pickup>
  <suggested_pickup_period>60</suggested_pickup_period>
  <location>Edwards AFB, CA</location>
  <station_id>KEDW</station_id>
  <latitude>34.91</latitude>
  <longitude>-117.87</longitude>
  <observation_time>Last Updated on Oct 19 2010, 9:55 am PDT</observation_time>
  <observation_time_rfc822>Tue, 19 Oct 2010 09:55:00 -0700</observation_time_rfc822>
  <weather>Mostly Cloudy</weather>
  <temperature_string>62.0 F (16.4 C)</temperature_string>
  <temp_f>62.0</temp_f>
  <temp_c>16.4</temp_c>
  <relative_humidity>100</relative_humidity>
```

Como você pode ver, cada trecho de informação está incorporado em uma tag. Por exemplo, a temperatura em Fahrenheit inicia e termina com a tag `<temp_f>`. Tudo que há entre as tags corresponde aos dados.

Primeiramente, você verifica se o caractere é um <. Se afirmativo, temos o início de uma tag:

```
if (inChar == '<') {
```

Se esse for o caso, você chama a função `addChar`, que verificará se o comprimento da string está dentro dos limites de `MAX_STRING_LEN`. Caso positivo, ela adiciona o caractere à sua string `tmpStr`. Essa função será analisada futuramente.

```
addChar(inChar, tmpStr);
```

Como você encontrou uma tag, `tagFlag` é definida como `true`, e `dataFlag` como `false`:

```
tagFlag = true;
dataFlag = false;
```

Se você alcançar o fim da tag (o caractere >):

```
} else if (inChar == '>') {
```

Então o caractere será adicionado à string `tmpStr`.

```
addChar(inChar, tmpStr);
```

Se você estiver atualmente processando uma tag e tiver alcançado seu fim, pode copiar a tag inteira de `tmpStr` (string temporária) para a string de tags (`tgrStr`). Para isso, você utiliza o comando `strncpy`:

```
if (tagFlag) {
  strncpy(tagStr, tmpStr, strlen(tmpStr)+1);
}
```

O comando `strncpy` copia parte de uma string para outra e requer três parâmetros: a string para qual você está copiando os dados, a string da qual os dados estão vindo e a quantidade de caracteres que devem ser copiados. Por exemplo, se você tivesse:

```
strncpy(firstString, secondString, 10);
```

Os primeiros dez caracteres de `secondString` seriam copiados em `firstString`. Em seu caso, você copia o conteúdo por inteiro, encontrando o comprimento da string temporária (`tmpStr`) mais um, e copiando essa quantidade de caracteres para a string de tags.

Assim que a string temporária tiver sido copiada, você deve limpá-la, deixando-a pronta para o próximo trecho de dados. Para isso, você chama a função `clearStr` e passa a ela a string que deseja limpar:

```
clearStr(tmpStr);
```

As duas flags são definidas como `false`, ficando prontas para receber as próximas informações:

```
tagFlag = false;
dataFlag = true;
```

Se o caractere lido for uma alimentação de linha (ASCII `10`):

```
} else if (inChar != 10) {
```

Você adiciona o caractere à string, caso esteja processando uma tag:

```
if (tagFlag) {
  addChar(inChar, tmpStr);
```

Você deseja ignorar as tags finais, por isso verifica se está processando uma tag e se atingiu seu final (comparando os dados com os caracteres de `endTag`):

```
if ( tagFlag && strcmp(tmpStr, endTag) == 0 ) {
```

Então, ignoramos a tag, limpamos a string e definimos as tags com seus valores-padrão:

```
clearStr(tmpStr);
tagFlag = false;
dataFlag = false;
```

O comando `strcmp` compara duas strings. Em seu caso, ele compara a string temporária (`tmpStr`) aos caracteres no array `endTag`:

```
strcmp(tmpStr, endTag)
```

O resultado será 0, se as strings corresponderem, e outro valor, se não corresponderem. Comparando seus dados com o array `endTag`, você verifica se nenhum dos três caracteres de tag final estão presentes.

Se a string atual for composta de dados:

```
if (dataFlag) {
```

Então, você adiciona o caractere atual à string de dados (`dataStr`).

```
addChar(inChar, dataStr);
```

Esse código verifica se você está processando uma tag e, caso afirmativo, armazena os caracteres na string de tags (`tagStr`). Se, por outro lado, estivermos trabalhando com dados, o código armazenará essa informação na string correspondente (`dataStr`). Dessa forma, ao final você terá a tag e os dados armazenados separadamente.

Se você encontrar um caractere de alimentação de linha, estará claramente no fim da string atual. Agora, deve verificar as tags para ver se representam os dados de temperatura, umidade ou pressão que lhe interessam.

```
if (inChar == 10 ) {
```

Para tanto, você utiliza a função `matchTag()` (da qual falaremos em breve), que verifica se a tag especificada está contida na string de tags e, caso isso ocorra, retorna um valor `true`. Você inicia procurando a temperatura na tag do valor em Fahrenheit (`<temp_f>`):

```
if (matchTag("<temp_f>")) {
```

Se afirmativo, você imprime a string de dados, a qual, caso a tag seja `<temp_f>`, deverá conter a temperatura em Fahrenheit.

```
Serial.print("Temp: ");
Serial.print(dataStr);
```

Em seguida, você verifica a temperatura em Celsius:

```
if (matchTag("<temp_c>")) {
  Serial.print(", TempC: ");
  Serial.print(dataStr);
}
```

A umidade:

```
if (matchTag("<relative_humidity>")) {
  Serial.print(", Humidity: ");
  Serial.print(dataStr);
}
```

E a pressão:

```
if (matchTag("<pressure_in>")) {
  Serial.print(", Pressure: ");
  Serial.print(dataStr);
  Serial.println("");
}
```

Então, limpamos todas as strings, deixando-as prontas para a próxima linha:

```
clearStr(tmpStr);
clearStr(tagStr);
clearStr(dataStr);
```

Fazemos o mesmo para as tags:

```
tagFlag = false;
dataFlag = false;
```

Em seguida, você tem suas funções de usuário. A primeira é a função que limpa as strings (`clearStr()`):

```
void clearStr (char* str) {
```

Essa função simplesmente encontra o comprimento da string passada utilizando o comando `strlen()`:

```
int len = strlen(str);
```

Depois, utiliza um loop `for` para preencher cada elemento do array com o caractere ASCII nulo (`0`):

```
for (int c = 0; c < len; c++) {
  str[c] = 0;
}
```

Em seguida, temos a função `addChar`. Como parâmetros, você passa a ela o caractere atualmente lido e a string atual:

```
void addChar (char ch, char* str) {
```

Você define dois novos arrays de caracteres e armazena neles as mensagens de erro:

```
char *tagMsg  = "<TRUNCATED_TAG>";
char *dataMsg = "-TRUNCATED_DATA-";
```

Caso você perceba que as strings ultrapassaram o comprimento de `MAX_STRING_LEN`, terá de substituí-las por essas mensagens de erro.

Agora, você verifica o comprimento da string para descobrir se ela atingiu o comprimento máximo:

```
if (strlen(str) > MAX_STRING_LEN - 2) {
```

Se isso ocorreu e estivermos atualmente processando uma tag:

```
if (tagFlag) {
```

Limpamos a string das tags e copiamos nela a mensagem de erro:

```
clearStr(tagStr);
strcpy(tagStr,tagMsg);
```

Se você estiver processando dados, então limpamos a string correspondente e copiamos para ela a mensagem de erro dos dados:

```
if (dataFlag) {
  clearStr(dataStr);
  strcpy(dataStr,dataMsg);
}
```

Limpamos a string e as tags temporárias:

```
clearStr(tmpStr);
tagFlag = false;
dataFlag = false;
```

Caso o comprimento da string não tenha excedido o máximo permitido, você adicionará à string o caractere atual lido e utilizará o comprimento da string para descobrir qual o último caractere (ou seja, o próximo ponto no qual você pode adicionar um caractere).

```
} else {
  // Adiciona um caractere à string
  str[strlen(str)] = ch;
}
```

Por fim, você alcança a função `matchTag()`, utilizada para verificar se a tag de pesquisa, passada a ela como parâmetro, foi encontrada ou não, retornando `true` ou `false`, respectivamente:

```
boolean matchTag (char* searchTag) {
```

A função é de tipo booleano, uma vez que retorna um booleano, e requer um array de caracteres como parâmetro:

```
  if ( strcmp(tagStr, searchTag) == 0 ) {
    return true;
  } else {
    return false;
  }
}
```

Alterando o URL do feed XML e as tags encontradas nele, você pode utilizar esse código para buscar dados em qualquer feed RSS. Por exemplo, poderia utilizar os feeds meteorológicos do Yahoo (*http://weather.yahoo.com*), navegando até a região que deseja analisar e clicando no botão RSS. O URL desse feed poderia ser, então, digitado no código. Você pode ver o código-fonte do feed clicando com o botão direito e escolhendo a opção correspondente. Dessa forma, é possível observar as tags e modificar o código para encontrar as informações mais relevantes.

Este último projeto mostrou-lhe como utilizar seu shield Ethernet para obter informações da Internet. Em projetos anteriores, você enviou dados do shield para fontes externas. Neste, você leu dados vindos da Internet. Em vez de exibir os dados meteorológicos na janela do monitor serial, você pode utilizar as habilidades que aprendeu nos projetos anteriores para representar essas informações em uma tela LCD ou em um display LED de matriz de pontos.

Resumo

Este capítulo final mostrou-lhe como conectar seu Arduino à Internet, seja para enviar dados na forma de uma página, um tweet para o Twitter, um e-mail, dados de sensores enviados ao Pachube, ou para solicitar uma página web e selecionar dados dela para uso próprio. Sabendo como conectar seu Arduino à Internet ou a uma rede LAN, você tem acesso a muitas novas opções de projetos. Dados podem ser enviados de qualquer ponto de sua casa, ou de seu escritório, em que haja uma porta Ethernet disponível; ou lidos da Internet para serem processados, exibidos ou utilizados pelo Arduino.

Você poderia, por exemplo, utilizar um feed meteorológico para determinar se está prestes a chover e emitir um aviso para que você recolha suas roupas do varal, ou feche uma clarabóia. As possibilidades do que pode ser feito com seu Arduino conectado à Internet são limitadas apenas por sua imaginação.

Assuntos e conceitos abordados no capítulo 17:

- como atribuir manualmente um endereço MAC e IP ao seu dispositivo;
- o conceito de cliente e servidor;
- como escutar conexões do cliente utilizando o comando `client.connected()`;
- como enviar código HTML para ser impresso por um cliente;
- como utilizar seu Arduino na função de servidor web;
- como conectar seu Arduino a um navegador;
- como verificar se os dados estão disponíveis, utilizando o comando `client.available()`;
- como ler dados com o comando `client.read()`;
- como enviar dados ao Pachube e visualizá-los como gráficos etc.;
- como enviar tweets para o Twitter.com a partir do Arduino, via proxy;
- como enviar e-mails a partir do Arduino;
- como capturar dados de um feed RSS e processá-los para exibição;
- como copiar strings com os comandos `strcpy` e `strncpy`;
- como comparar strings com o comando `strcmp`;
- como preencher locais da memória com o comando `memset`;
- como encontrar o tamanho dos arrays com o comando `sizeof`;
- como processar strings com o comando `sprintf`;
- como descobrir o comprimento de strings com o comando `strlen`;
- como pesquisar substrings com o comando `strstr`;
- como descobrir um endereço IP com o comando `ping`;
- como verificar se um cliente está conectado utilizando o comando `client.connect()`;
- como criar uma conexão Ethernet com o comando `Ethernet.begin(mac, ip)`;
- como criptografar, em base 64, nomes de usuário e senhas usando utilitários de sites;
- como utilizar a biblioteca `EthernetDHCP.h` para atribuir automaticamente um endereço IP;
- como utilizar os comandos `post` e `wait` da biblioteca Twitter;
- como pesquisar tags em um feed XML.

Símbolos

A

B

C

D

E

F

M

N

O

P

Q

R

S

T

U

V

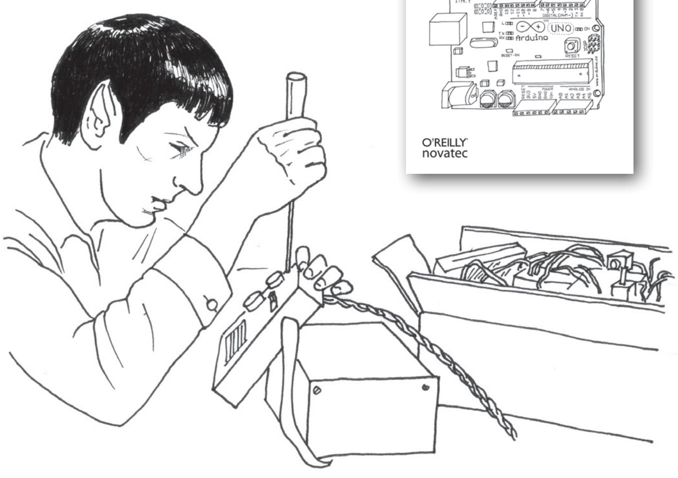
W

X

Conheça também

O livro *Primeiros Passos com o Arduino* adota uma abordagem muito prática. Para utilizar os exemplos introdutórios deste guia, tudo que você precisa é de um Arduino Uno (ou modelo anterior), um cabo USB A-B e um LED. O download do ambiente de desenvolvimento do Arduino pode ser feito gratuitamente.





Neste livro, você aprenderá sobre:

- Design de interação e computação física
- O hardware do Arduino e seu ambiente de desenvolvimento de software
- Noções básicas de eletricidade e eletrônica
- Prototipagem em uma protoboard sem solda
- Elaboração de diagramas esquemáticos

Junte-se às centenas de milhares de projetistas que descobriram essa incrível (e educativa) plataforma. Escrito pelo cofundador do Arduino Project, *Primeiros Passos com o Arduino* leva você para dentro dessa diversão.